

Scope (قلمرو، حوزه): حوزه‌ی یک متغیر محدوده‌ای از دستورات است که متغیر در آن محدوده

قابل مشاهده باشد. یک متغیر در یک دستور قابل مشاهده است اگر بتواند در آن دستور مورد

ارجاع قرار گیرد. متغیر در یک واحد برنامه یا یک بلوک، محلی محسوب می‌شود، اگر در همان قسمت

اعلان شده باشد. متغیرهای غیرمحلّی، متغیرهای هستند که در آن واحد برنامه یا یک بلاک

قابل مشاهده هستند، اما در آن قسمت اعلان نشده اند. متغیرهای سراسری نوع خاصی از

متغیرهای غیرمحلّی هستند. از نظر Scope زبان ها به ۲ دسته تقسیم بندی می شوند:

۱- حوزه ایستا (static scope): الف) زیربرنامه ای خود را تو دارند: زبان هایی مانند

Ada, JavaScript, Common Lisp, Scheme, Fortran 2003+, F#

مثال JS

python. اگر ارجاع به متغیر در زیربرنامه ای sub1

وجود داشته باشد و اعلان در sub1 پیدا شود (وجود function sub1())

داشته باشد. x محلی است. اگر اعلانی برای x و x = 5 و x = x + 1 و sub2() و x ← big.x is hidden

در sub1 وجود نداشته باشد، x غیرمحلّی است و function sub2()

مستقیم در زیربرنامه ای که زیربرنامه ای sub1 non-local و y = x و 3

در آن تعریف شده است، ادامه میابد. این زیربرنامه var x = 3 و

والد ایستا (static parent) نامیده می شود. اگر // local و x++ و sub1()

اعلای در آنجا نیز پیدا نشود، مستقیم در والد والد ادامه میابد. (به طور کلی مستقیم در اجداد ایستا

انجام می شود. در مثال منتهی قبل، ارجاع به متغیر در Sub2 را در نظر بگیرید. نحوه اعلانی

برای در Sub2 وجود ندارد، غیر محلی است. بستگی در والدین Sub2 یعنی big

انجام می شود و در حقیقت، دسترسی به متغیر در اعلان شده در big دارد.

متغیر مخفی: برخی از اعلان ها ممکن است در قسمت هایی از کد مخفی شوند (دیگر قابل

دسترسی نباشند). برای مثال در Sub هر ارجاع ساده ای به در حقیقت، دسترسی به همان

محلی است. بنابراین در اعلان شده در Sub1 (big) مخفی شده است. زبان

JS اجازه دسترسی به متغیر مخفی را نمی دهد؛ اما زبان Ada ^{اجازه} می دهد به متغیرهای مخفی شده ی

موزه ی بالاتر با آوردن نام موزه ی مورد نظر، دسترسی داشت (big).

ب. زیر برنامه ی تو در تو قرار نده: اگر شما بنویسیم، یعنی در
Sub () (تعریف نمی شود)

محلی. اگر :: بنویسیم، یعنی در بالای برنامه (مخفی).
:

در C++، اجازه ی دسترسی به متغیر مخفی داده می شود.

main ()

{

int x;

x :: 2

} scope

۲- حوزه ی پویا (Dynamic scope): حوزه ی متغیرها در زبان های مثل APL.

Shobol 4: نسخه های اولیه ی Lisp از نوع پویا است. حوزه ی پویا به پای مبتنی بوده بر رابطه ی

مکانی زیر برنامه ها، مبتنی بر دنباله ی فراخوانی زیر برنامه ها است. بنابراین فقط می تواند در

زبان اجرا تعیین شود. $x = 5$ در Sub2: Sub2 $\rightarrow$ Sub1 $\rightarrow$ big

scope و life time: وقت شودی گاهی scope و life time یک متغیر به هم وابسته

می باشند. متغیر محلی در یک متد زبان Java را در نظر بگیرید. حوزه ی این متغیر همان بلاک

می باشد. طول عمر این متغیر از زمان ورود به متد تا زمان خاتمه ی اجرای آن است. در واقع

حوزه، یک مفهوم مکانی و طول عمر یک مفهوم زمانی است که به یکدیگر وابسته می باشند.

به شرطی که از متد، متد دیگری را صدا نزده باشیم. می تواند بین scope و life time

چنین ارتباطی وجود نداشته باشد. مثلاً در همان متد قبلی، متغیر را از نوع static تعریف می کردیم

یا از متد قبلی متد دیگری را فراخوانی می کردیم.

```
sub()
{
    int x;
    ==
}
main()
{
    int x;
    ==
    sub();
    ==
}
```


انواع داده

۱- انواع داده‌ی اولیه :

(I) نوع داده‌ی عددی (الف) نوع داده‌ی صحیح : مقادیر صحیح را می‌فاهیم ذخیره کنیم، پیاده‌سازی `int` و `integer`

سخت افزاری، نیاز به شبیه‌سازی نرم افزاری نداریم (اما در بعضی مواقع نیاز هست. مثل

`long integer` در پایتون و `F#`. مثال: `L ۶۷۸۹۹ ۵۶۰۹۰۵ ۱۷۱۲۳`

بسیاری از کامپیوترها از اعداد صحیح با اندازه‌های مختلف پشتیبانی می‌کنند. اغلب نوع داده‌ی

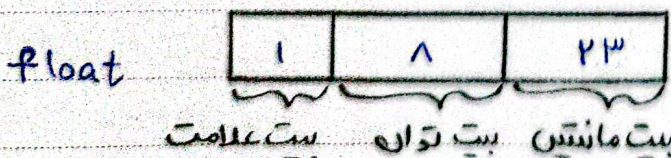
صحیح در زبان‌ها، به‌طور مستقیم توسط سخت‌افزار پشتیبانی می‌شود. برخی از انواع صحیح به‌طور

مستقیم توسط سخت‌افزار نمی‌توانند ارائه شوند؛ مثل نوع داده‌ی صحیح بلند در پایتون یا `F#` که

می‌تواند طول عدد محدود داشته باشد. نوع داده‌ی صحیح، اعداد منفی را نیز می‌تواند نگهداری

کنند که برای اعداد منفی از روش مکمل ۲، استفاده می‌شود.

(ب) نوع داده‌ی معین شناور (Floating point) : استاندارد IEEE ۷۵۴.



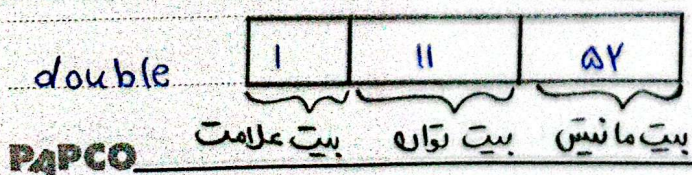
برای نگهداری $۰,۸ \times ۲ = ۱,۶$

عدد $۰,۸$ باید $۰,۶ \times ۲ = ۱,۲$

پایانی که اعشار آن $۰,۲ \times ۲ = ۰,۴$

به خودش برسد، آن $۰,۴ \times ۲ = ۰,۸$

را ضرب در ۲ کنیم.



مانتیس

$$0.8 = 1.0011001100 \times 2^{127+126} = 1.0011001100 \times 2^{253} \rightarrow \text{توان}$$

$$0.2 = 0.0011001100110011 \times 2^{-127+126} = 0.0011001100110011 \times 2^{-1} \rightarrow \text{توان}$$

در این استاندارد، باید جای ممیز را عوضی کنیم و به جای ببریم که یک «۱» پشت ممیز باشد که

یک عدد نرمال به دست بیاید. پس به توان ۲، در صورت float بودن عدد ۱۲۷ و در صورت

double بودن عدد ۱۰۲۳ را اضافه می کنیم و آن عدد را به عنوان توان ذخیره می کنیم. همچنین

منبر ۲ کردن را تا زمانی ادامه می دهیم که بیت های لازم پر شود. هیچ عددی نمی توان

به صورت دقیق به صورت float یا double ذخیره کرد. (عدد خیلی نزدیک عدد مذکور است.)

پس برای کاربردی مثل ذخیره ی فیزیک مناسب، اما برای کاربردی تجاری مثل ذخیره ی رمز ارز

نامناسب است.

پایه ی داده ی مطلق: در برخی از زبان ها مانند فورترن و پایتون نوع اعداد مطلق پشتیبانی

می شود. به صورت یک هفت مرتب از مقادیر ممیز شمار و نمایش داده می شود. در زبان پایتون

قسمت موهومی یا عرفی از فکت می شود. مثال $(7 + 3j) + a + bj$

a	1	1	۲۳
b	1	1	۲۳

نوع داده‌ی ممیز ثابت (fixed point) : در پردازش داده‌های بکاری، نوع داده‌ی

ممیز ثابت، یک نوع داده‌ی اساسی محسوب می‌شود. چراکه می‌تواند عدد را به طور دقیق

نمایش دهد. در نتیجه زبان‌هایی مانند کوپول، C# و F# دارای نوع داده‌ی ممیز ثابت

می‌باشند. ۹۹۹۹ picture X : در کوپول (۲، ۲) (decima) در C#
فاصله دارد

کد BCD : به هر رقم، یک کد اختصاصی می‌دهد (ارقام ۰ تا ۹).

۰ → ۰۰۰۰ و ۱ → ۰۰۰۱ و ۲ → ۰۰۱۰ و ۳ → ۰۰۱۱ و ۴ → ۰۱۰۰ و ۵ → ۰۱۰۱ و ۶ → ۰۱۱۰ و ۷ → ۰۱۱۱ و ۸ → ۱۰۰۰ و ۹ → ۱۰۰۱

نوع ممیز ثابت فعلی شبیه رشته‌های کاراکتری ذخیره می‌شود. به این صورت که برای هر رقم

دهدی از یک کد دو بیتی استفاده می‌شود (نمایش BCD). در برخی از موارد هر رقم در یک

بایت ذخیره می‌شود، و در برخی از موارد دیگر، هر دو رقم در یک بایت فشرده می‌شود.

نوع داده‌ی boolean : شاید یکی از ساده‌ترین انواع داده باشد؛ چراکه تنها دارای دو

مقدار است : true (درست) و false (نادرست). نوع داده‌ی بولین در زبان ALGOL 6۰

معرفی شد. (static scope نیز در این زبان معرفی شد). و در اغلب زبان‌های بعدی

مورد استفاده قرار گرفت. یک مقدار بولی را می‌توان تنها با استفاده از یک بیت نمایش داد.

اما از آنجایی که یک بیت در حافظه به شکل ۱ قابل دسترسی نیست، این نوع داده اغلب در واحدهای بزرگ تر قابل آدرس دهی (معمولاً یک بایت) ذخیره می شود.

(ج) **نوع کرکتری**: داده های کرکتری به صورت کدهای عددی در کامپیوتر ذخیره می شوند.

مداول ترین روش کوگذاری کد ۱ بیتی، آسکی است. با توجه به جهانی شدن تجارت و نیاز به ارتباط

کامپیوترها با یکدیگر در سراسر جهان، مجموعه ای کرکترهای آسکی کافی به نظر نمی رسد. مجموعه

کرکتر یونی کد که یک مجموعه کرکتر ۱۶ بیتی است معرفی شد. Java اولین زبانی بود که به طور گسترده ای از مجموعه کرکترهای یونی کد استفاده نمود. پس از آن در زبان های مانند C++

گسترده ای از مجموعه کرکترهای یونی کد استفاده قرار گرفت. برخی از استانداردهای یونی کد: C#, F#, Perl و پایتون مورد استفاده قرار گرفت. برخی از استانداردهای یونی کد:

UCS-2 (در سال ۱۹۹۱) UCS-4 (یا همان UTF-32) 'A' 'a'

(II) **رشته های کرکتری**: چند عملیات برای کار با رشته های کرکتری وجود دارد: انتخاب، عملگر

الحاق (برای اتصال دو رشته)، انتخاب زیر رشته (برای جدا کردن قسمتی از یک رشته)، مقایسه ای

رشته ها (بر اساس ترتیب دیکشنری انجام می شود). حرفی که زودتر در دیکشنری آمده باشد، کوچکتر

است. مثال: "ro.z" < "re.za"، تطبیق الگو، طول رشته. (دستورات در زبان C است.)

در C ندارد strlen

streq

strcat

strcmp

PAPCO

عبارت منظم (RE): تعریف بازگشتی آن این است که هر عضو آلفابت، یک عبارت منظم است.

رشته‌ی به طول صفر (λ) یک عبارت منظم است. اتصال دو عبارت منظم، خودشان یک عبارت منظم است.

هر تعداد تکرار (*) هر عبارت منظم، خود یک عبارت منظم است. اگر a منظم باشد، a^+ هم منظم است. شامل صفر از تکرار a^* بالا

تطبیق الگو: زبان‌هایی مانند Perl، Ruby، JS، PHP دارای عملگرهایی جهت تطبیق Pattern Matching

الگوی باشند. برای تعریف الگو از عبارت منظم استفاده می‌شود. اولین زبانی که تطبیق الگو

در آن معرفی شد ۴- shobal بود.

ααα

مثال * می‌فهمیم الگویی را تعریف کنیم که رشته‌هایی را بپذیرد که با رقم شروع نمی‌شود. (از

زبان Perl استفاده شود). $/[A-Z a-z][A-Z a-z \backslash d]^*/$ هر تعداد تکرار (رقم (digit)

این رشته‌ها با یک حرف شروع می‌شوند و با یک یا چند حرف یا عدد ادامه پیدا می‌کنند.

مثال * می‌فهمیم الگوی رشته‌های عدد را تعریف کنیم. (Perl) این الگو باید این رشته‌ها را

بپذیرد: 1، 14، 14.۵، ۵3، ۵3.۵، ۵3.۵3، 12.۵3، 123

اما ممیز آنها (.) را نباید بپذیرد. $/\backslash d^* \cdot ? \backslash d^*/$

اگر اینطوری بنویسیم، ممیز آنها را هم می‌پذیرد.

جواب درست:

/ \d * \. ? \d + | \d + \. /

انواع رشته ها از نظر طول رشته

پیدا بین انتخاب در رابطه با مقادیر طول رشته وجود دارد که عبارت اند از:

۱- رشته با طول ثابت (static length): طول رشته می تواند ایستا باشد و در زمانی ایجاد

رشته مشخصی شود. برای مثال رشته در زبان فور ترن از این نوع است. یا مثلاً کتابخانه ی

.net در C# و F#. این رشته ها همیشه پُر اند. $\text{char}(20) \rightarrow \text{Reza}$ (بقیه و space میخوره)

۲- رشته با طول پویای محدود شده (دارای صوابلا): در این حالت اجازه می دهیم که

طول رشته متغیر باشد. دارای یک مقدار حداکثر است که در تعریف متغیر مشخص می شود.

مانند رشته ها در زبان C و C++ که می توانند هر تعداد کرکری را بین و صوابلا ذخیره

می کنند. $\text{varchar}(20)$

۳- طول رشته بدون هیچ محدودیتی قابل تغییر باشد (رشته با طول پویا): مثل زبان های

JS، Perl، تخصیصی و آزاد سازی حافظه ی پویا در صیه اجرا؛ چرا که محدودیتی ندارد.

نکته* برخی زبان ها هر سه نوع رشته ی بالا را دارند. مثل زبان Ada.

پیاده سازی نوع داده‌ی رشته‌ی کرکتری

رشته‌ی طول ایستا
طول
آدرس

۱- رشته با طول ایستا : با استفاده از توصیفگر زبان کامپایل .

توصیفگر زبان کامپایل

رشته با طول پویای محدود شده
حداکثر طول
طول فعلی
آدرس

۲- رشته با طول پویای محدود شده (داینامیک) :

رشته‌های پویای محدود شده در C و C++ که توصیفگر

زبان اجرا ندارند، اولاً انتهای رشته را با کرکتر `null` توصیفگر زبان اجرا

مشغولی کنند و ثانیاً چون حداکثر طول را ندارند، هنگام ارجاع به عناصر رشته، کنترل باره

برای اندیس نمی‌توان انجام شود.

۳- رشته با طول پویا : به روش‌های زیر می‌توان پیاده‌سازی نمود :

الف) استفاده از یک لیست پیوندی به طوری که در زمان افزایش طول رشته، خانه‌های مورد نیاز

جدید از حافظه‌ی heap تخصیص میابد.

ب) ذخیره‌سازی کامل رشته در خانه‌های مجاور حافظه : اگر رشته نیاز به افزایش طول داشته

باشد و در خانه‌های مجاور، حافظه وجود نداشته باشد، باید قسمت جدیدی از حافظه که بتواند

یک گراندا این روش پیاده‌سازی این است که هر انفر که پشت سر هم می‌بیند

باید پشت سر هم نیز ذخیره کند.

PAPCO


```
graph LR
    A["14 | r | e | z |"] --> B["a | s | h | e |"]
    B --> C["i | b | a | n |"]
    C --> D["i | | | | o"]
```

بسیار خانه های حافظه ای مربوط به رشته قلبی، آزاد می شود. new → rezaSheibani

الف) نوع داده‌ی شمارشی: نوع داده‌ی شمارشی، مقداریرسمکن عبارت انداز ثابت‌های دارای

نامی که توسط user (برنامه نویسنده) مشخص شده و تعیین می شوند. به مثال زیر در نرم افزار

enum days = { Mon, Tue, Wed, Thu, Fri, Sat, Sun }; # دقت کنید

در این مثال داده ها ترتیبی هستند. عملگرهایی که برای days d

Predecessor: عبقري قاتل

قابل تعریف هستند: انتساب، رابطه‌ای $(\prec, \succ, \leq, \geq)$ ، succ ، pred

Successor: علفير علفرى رافىر

عموماً به طور فزاینده‌ای، مقادیر صحیح مصرف آب، ... به تأییدهای شما، Shi sun to Mon

انتخاب داده می شود. افزایش خوانایی و افزایش قابلیت اعتماد، از مزایای نوع داده ی

بنمایا رشتی است. آخر اینش خوانایی به این دلیل است که مقدار نام گذاری شده به راحتی قابل

تشفیعې اند. د خصوصي افزایش قابلیت اعتماد، د زمان هابی مانند Ada، C#، F# و

Java دو مورد زیر قابل ذکر است:

۱- هیچ گونه عملیات حسابی بر روی مقادیر شمارشی مجاز نیست. به عنوان مثال در

صفحه ی قبل، از جمع روزهای هفته جلوگیری می شود.

۲- نمی توان مقدار خارجی از مقادیر تعریف شده به متغیر شمارشی نسبت داد.

نکته* با توجه به اینکه در زبان C با متغیرهای شمارشی، مشابه متغیرهای مدیف شمارشی شود.

این رومزیت ذکر شده را ارائه می دهد.

در زبان Ada: `type days is (Mon, Tue, Wed, Thu, Fri, Sat, Sun);`

نکته* در واقع آبجکت d یک مقدار دارد. (با آرایه اشتباه نشود.)

ب) نوع داده ی زیر قلمرو: دنباله ی پیوسته ای از یک نوع داده ی ترتیبی است. در زبان

پاسکال، معرفی شد و در زبان های مانند رلنی و Ada به طور گسترده استفاده شد.

در زبان Ada، نوع داده ی زیر قلمرو را به صورت زیرنوع (subtype) تعریف می کنیم که

در واقع نام جدیدی برای نسخه ی محدود شده ی ابرنوع (parent type / supertype)

محسوب می شود.

در زبان Ada: $\text{subtype weekdays is days range Fri..Sun}$ ^{نوع والد}

در زبان Ada: $\text{subtype index is integer range 1..12}$

مثال* در یک فروشگاه می توانیم میزان فروش هر روز را در آرایه ای که اندیس آن، همان



روز است، ذخیره کنیم. این برنامه را با زبان Ada بنویسید.

نوع داده ی جدید $\text{type sales is array (weekdays) of integer}$ ^{نوع عناصر آرایه} ^{نوع اندیس آرایه}

تمام عملیات تعریف شده برای نوع والد، برای زیرنوع نیز تعریف می شود. یکی از متداول ترین

استفاده ها از نوع داده ی ترتیبی تعریف شده توسط کاربر، استفاده از آرایه های اندیس آرایه (تعریف)

است. با توجه به اینکه متغیرهای زیرقلمرو، شرایطی توانند دربارگی مشخصی شده ذخیره

شوند، باعث بهبود توانایی می شوند؛ چراکه انتساب مقداری خارج از بازه، فظا در نظر گرفته

می شود و این می تواند قابلیت اعتماد را نیز افزایش دهد.

پس نوع داده ی آرایه: آرایه مجموعه ای از عناصر هم نوع است، به گونه ای که موقعیت هر

عنصر مشخصی، نسبت به اولین عنصر تعیین می شود. ارجاع به عناصر مورد نظر در آرایه، با

استفاده از اندیس انجام می شود. در تعریف نوع داده ی آرایه، دو نوع مستقل وجود دارد:

نوع اندیس که می تواند هر نوع داده ی ترتیبی مانند `bool`، `کرکتری`، `شمارشی` و `زیر قلمرو`

در زبان `Ada` استفاده بشود. `رتبوی نوع عناصر آرایه` که این هم می تواند هر نوع داده ای

داشته باشد. `type list is array (index) of float;`
↓
`days`

نکته * در برخی از زبان ها، `درایین بازه ی اندیس` به صورت `فمنی مشمنی` می شود. برای مثال

در زبان های `متنی بر C`، `درایین تمام بازه ی اندیس ها`، `منفر است`. در زبان `فورترن`، به

صورت `پیشمنی درایین یک` می باشد.

در زبان های `برنامه سازی`، یا `توجه به مقید سازی اندیس` و `مقید سازی حافظه`، `آرایه ها` می توان

به `دسته تقسیم بندی` نمود که عبارت اند از:

۱- `آرایه ای ایستا (static array)`: `آرایه ای است که در آن تعیین باره ی اندیس و`

`تعیین حافظه به صورت ایستا (قبل از اجرا) انجام می شود.`

این مثال در زبان `C` است و اندیس های این `آرایه از` `static int list[10]` و

و... است و `life time` این `متغیر قبل اجرا شروع می شود` و `مقید سازی حافظه و اندیس نیز همینطور`

۲- `آرایه ی پویای پشته ای ثابت (Fixed Stack-Dynamic Array)`: `آرایه ای`

است که باره‌ی انریس به صورت ایستا مقید می‌شود. اما مقیدی حافظه در صحن اجرا در زمان

fun ()

در صحن به اعلان متغیر (در زمان ورود به بلاک) انجام می‌شود.

```
int list [10];
...
}
```

۳- آرایه‌ی پویا (stack - Dynamic Array):

آرایه‌ای است که هم مقید سازی انریس و هم مقید سازی

main ()

حافظه در زمان اجرا انجام می‌شود (به صورت پویا). و

{

fun ()

پس از اینکه باره‌ی انریس هم مقید شد و حافظه‌ی لازم

مقیدی داده شد، دیگر این موارد در طول عمر متغیر ثابت باقی می‌ماند. مزیت آن نسبت به

نوع دوم این است که قابلیت انعطاف بیشتری دارد؛ چرا که تا زمانی استغاده از آرایه، نیازی

به مقیدی کردن اندازه‌ی آرایه و مقید سازی انریس نیست. به مثال زیر در زبان Ada وقت کنید:

Get (list-len);

declare

list : array (1..list-len) of integer;

begin

≡

end;

نوع انریس }
مقید سازی انریس }
نوع هر عنصر }
مقید سازی حافظه }

۴- Fixed heap dynamic array (آرایه‌ی پویای هری ثابت): بازه‌ی اندیس‌ها

و حافظه، پس از تخصیص دیگری تغییر نمی‌کند. یعنی در زمان درخواست کاربرد در همین اجرای

برنامه، مقداری شود که باعث افزایش قابلیت انعطاف برنامه نویسی می‌شود. به جای پیوسته

حافظه‌ی لازم از هم تخصیص میابد. زیرا برنامه نویس با توجه به نیازش درصورت اجرا برایش حافظه

گرفته می‌شود. `int * p = new int [20];`

۵- آرایه‌ی پویای هری (heap dynamic array): آرایه‌ای که تقدر بازه‌ی اندیس

و تخصیص حافظه به صورت پویا انجام می‌شود و این دومورد به دفعات می‌تواند در طول عمر متغیر

تغییر کند. برای مثال به آرایه‌ی پویای هری که به صورت عمومی (کلی) در زبان C# تعریف

شده، توجه کنید: `list<string> mylist = new list<string>();`

`mylist.add("reza");`

`mylist.remove("ali");`

پایه سازی آرایه

اندیس این آرایه از نوع زیرقلمرو

این مثال در زبان پاسکال است: `var L: array [-5..5] of real;`
 LB UB type

و نوع داده‌ی real، ۶ بیتی است. Low boundary up boundary

(از آنجا که سفت افزار کامپیوتر، داده‌های ۴ بیتی و ۱ بیتی را پشتیبانی می‌کند، نتیجه

Subject:
Date:

توصیفگر

LB
UB
type
آدرس α

می گیریم که اینها یک تشبیه سازی نرم افزاری انجام می شود. این امر

باعث می شود که سرعت پایین بیاورد؛ زیرا از منابع سخت افزاری نمی توان

به طور مستقیم استفاده کرد. ← سؤال پایانی دوم *

LB
LB+1
⋮
I →
مثلاً 3
UB

$$L_value(L[I]) = \alpha + (I - LB) \times E \rightarrow \text{size of (type)}$$

$$= \alpha - LB \times E + E \times I = v_0 + E \times I$$

$$\text{آدرس خانه اندیس صفر} = v_0 = L_value(L[0]) = \text{مبدأ مجاری}$$

نکته ی امکانی * مبدأ مجاری در توصیفگر ذخیره می شود و مرتب آن، افزایش سرعت دسترسی به عناصر آرایه.

M : array [1 ... 3 و 1 ... 4] of integer;

LB₁ UB₁ LB₂ UB₂

v ₀
LB ₁
UB ₁
LB ₂
UB ₂
type
آدرس

1	2	3	4
7	8	9	10
-1	-2	-3	-4

م [3 و 2]

E : size of (type) اندازه خانه

$$S : (UB_2 - LB_2 + 1) \times E$$

اندازه هر سطر

$$L_value(M[i, j]) =$$

$$\alpha + ((i - LB_1) \times (UB_2 - LB_2 + 1) + (j - LB_2)) \times E$$

تعداد عناصر هر سطر سطر مورد نظر چند خانه جلو آمدیم

$$= \alpha - LB_1 \times S - LB_2 \times E + i \times S + j \times E = v_0 + i \times S + j \times E$$

v₀

همیشه ثابت است

$$v_0 = L_value[0, 0]$$

(r_a)

یا سوره مورد استفاده قرار گرفت. به مثال زیر در زبان یا سوره دقت کنید:

$$\text{mat} = \begin{bmatrix} [1, 1, 1], [1, 0, 1], [1, 1, 1] \\ [1, 0, 1], [1, 0, 1], [1, 0, 1] \\ [1, 1, 1], [1, 0, 1], [1, 1, 1] \end{bmatrix}$$

۱۱۹

(ت) نوع داده‌ی رکورد (struct): مجموعه‌ای از عناصر داده‌ای می‌باشد که هر یک از عناصر

دارای یک نام می باشند و می توانند نوع و اندازه‌ی متفاوت داشته باشند. در اواخر دهی ۶۰

در زبان C، C++ و C#، کوردها به نوع داده‌ی struct پشتیبانی می‌شوند. تفاوت اصلی

بلکه با استفاده از نام عناصر استفاده می‌شوند.

مثال * در زبان Ada. type Employee - Name - Type is record
first: string (1..20); //offset=0
middle: string (1..10); //offset=20
Last: string (1..20); //offset=30
end record;

تعریف یک آرایه: N: Employee - Name - Type;

N.first = "reza"; مقداردهی

از نقطه برای ارجاع به عنصر

$$offset(A_i) = \sum_{j=1}^{i-1} sizeof(A_j)$$

فاصله تا ابتدای رکورد

عین الله	آدرس
جعفر نژاد	
قمری	

نکته * عناصر رکورد در فایلهای بایت سرهم حافظه ذخیره

می شود. (مربوط به پیاده سازی رکوردها)

آفست عنصر، آرم رای توان در زمان کامپایل حساب کرد.

عناصر اول
عناصر دوم
عناصر ۲م

رکورد
نام عنصر
نوع
آفست
نام عنصر
نوع
آفست
نام عنصر
نوع
آفست
آدرس

ش (نوع داده ای رکورد یا طول متغیر (variant record یا union): رکورد متغیر، رکوردی
در زبان های خانواده C

است که متغیرهای آن ممکن است در طول اجرای برنامه، انواع مختلفی را ذخیره کنند. برای
مثال کارمندی که ممکن است حقوق خود را به صورت ساعتی یا ماهیانه دریافت کند. مناسب
است که این نوع رکورد، با شکل های متفاوتی تعریف نشود. چنین رکوردی می تواند دارای یک
یا چند عنصر باشد که بین شکل های مختلف، یکسان هستند.

مثال * در زبان Ada. type shape is (circle , Triangle , Rectangle);
 type colors is (red , Green , Blue);
 type Figure (Form : shape) is
 record
 Filled : Boolean ;
 color : colors ;
 case Form is
 when circle =>
 Diameter : Float ;
 when Triangle =>
 Left_side : integer ;
 Right_side : integer ;
 Angle : Float ;
 when Rectangle =>
 side_1 : integer ;
 side_2 : integer ;
 end case ;
 end record ;

در توصیف‌گر، خانه‌هایی که
 چند نام دارند، در واقع در یک
 آدرس قرار دارند. پس به آنها
آلیاس گفته می‌شود.

switch-case مثل when circle =>

در زبان C++

Diameter : Float ;

when Triangle =>

Left_side : integer ;

Right_side : integer ;

Angle : Float ;

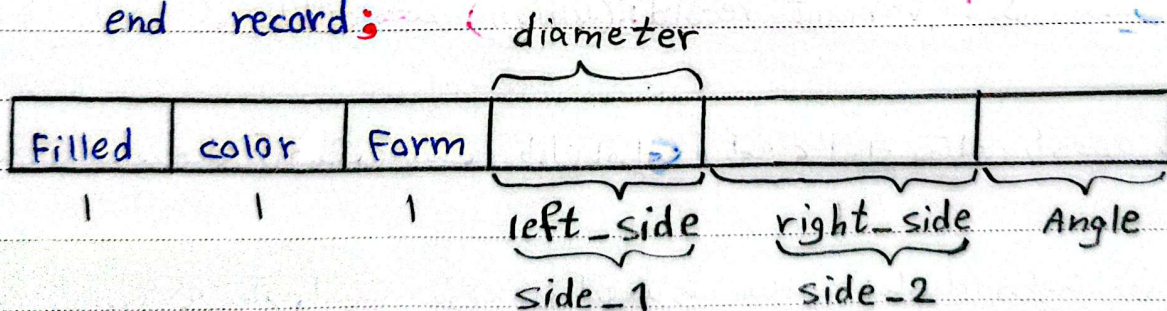
when Rectangle =>

side_1 : integer ;

side_2 : integer ;

end case ;

end record ;



(ج) نوع داده‌ی لیست: لیست‌ها برای اولین بار در زبان برنامه‌نویسی تابعی lisp معرفی شدند.

در زبان‌هایی مانند scheme نیز مورد استفاده قرار گرفتند. در زبان lisp، لیست‌ها در یک

زوج پرانتز تعریف می شوند و می توانیم لیست های تودرتو داشته باشیم:

نام تابع

۳ عنصر که یکیش خودش لیست است $\rightarrow (A (B C) D)$ عنصر $\rightarrow (A B C D)$
پارامتر

در زبان lisp، تمامی دستورات نیز به صورت یک لیست نوشته می شوند.

در واقع head لیست یا عنصر اولیهی لیست، تابعی است که باید بر روی آرگومان ها (پارامترها)

مراجعه شود. دو تابع معروف زبان lisp عبارت اند از تابع CAR که اولین عنصر از لیست

پارامترها را برمی گرداند: $(A B C)$ را برمی گرداند $(CAR (A B C))$
نشان دهندهی داده نام تابع

و تابع CDR که لیست پارامترها غیر از اولین عنصر را برمی گرداند:

لیست $(B C)$ را برمی گرداند $(CDR (A B C))$

لیست $(B C)$ را برمی گرداند $(CAR (CDR (A (B C) D)))$

پایه سازی لیست ها با لیست پیوندی انجام می شود و ساختار dynamic دارد.

(چ) نوع داده ای اشاره گر (reference، pointer): اشاره گر، نوعی است که متغیرهای آن

حاوی آدرس حافظه و یک مقدر ویژه به نام null (nil) می باشد. اشاره گر ها معمولاً جهت دسترسی

به متانی از حافظه به نام heap که در صیه اجرا به صورت پویا تخصیص میابد، استفاده می شود.

یادآوری: به متغیرهایی که به صورت پویا از heap تخصیص میابد، متغیرهای پویای هری می گوئیم که دارای شناسه نیستند (انیم ندارند) و تنها از طریق اشاره گر مورد استفاده قرار می گیرند. برای همین به آنها متغیرهای بی نام می گوئیم.

مشکلات مربوط به اشاره گرها

اشاره گر سرگردان (اشاره گر معلق، Dangling pointer): اشاره گری است که حاوی آدرس متغیر پویای هری ای است که حافظه ی آن آزاد شده باشد. اشاره گر سرگردان مشکل بسیار است درست می کند؛ چرا که به مکانی اشاره می کند که دوباره به یک متغیر پویای هری جدید، تخصیص داده شده باشد. دنباله ی عملیات زیر، اشاره گر معلق ایجاد می کند:

int * p

int * q

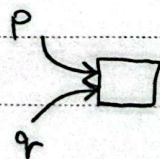
p = new int

q = p

:

delete (p)

الف) یک متغیر پویای هری جدید، ایجاد شود و اشاره گر



p به آن اشاره کند.

ب) اشاره گر p به اشاره گر q انتساب داده شود.

→ null

delete (p)

ج) متغیر پویای هری که p به آن اشاره می کند، به صورت صریح آزاد شود و p، null شود.

اما چون q تغییر نمی کند، با این وضعیت q یک اشاره گر معلق محسوب می شود.

متغیر پویای هدری گم شده (زباله، Garbage، Lost heap-dynamic variable):

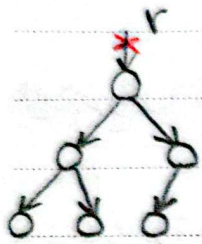
متغیری است که از heap به آن حافظه تخصیص داده شده،

`int* p;`

اما دیگر به آن دسترسی نداریم و زباله تولید شده. دستورات

`p = new int;`

مقابل، زباله تولید می کنند:



مثال دیگر این است که ما به اشتباه، ریشه ی یک درخت را از دست بدهیم.

مثلاً به جای که بنویسیم $r' = r$ بنویسیم $r = r'$

در این حالت، شما اشاره گری که از طریق آن به درخت دسترسی داشتیم، از بین رفته و کلی درخت به

زباله تبدیل می شود.

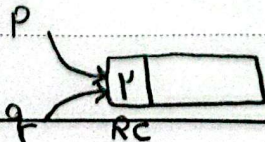
رویه های حل مشکلات اشاره گری

۱- مشتاق (eager): شمارنده ی ارجاع (Reference counter): برای هر عنصر

heap، فضای اضافی برای شمارنده ی ارجاع، منظور می شود. شمارنده ی ارجاع مشتقی می کند

که چند اشاره گر به آن عنصر اشاره می کنند. وقتی که عنصر از لیست فضای آزاد، تخصیص

میابد، شمارنده ی ارجاع آن یک است. هر وقت اشاره گری به آن اضافه شود، یک واحد به شمارنده



اضافه و اگر اشاره‌گری از آن برداشته شود، یک واحد شمارنده کم می‌کنیم. می‌توان با استفاده

از آن، از بروز اشاره‌گر سرگردان و زباله جلوگیری کرد؛ چراکه با کم شدن یک واحد شمارنده‌ی

ارجاع اگر شده باشد که آن عنصر می‌تواند به لیست فضای آزاد برگردانده شود و اگر

نشود، مشخص است که اشاره‌گرهای دیگری دارند به آن اشاره می‌کنند و نمی‌تواند آزاد شود.

مهم‌ترین مشکل شمارنده‌ی ارجاع، هزینه‌ی نگهداری آن است؛ چراکه باید دستور انتساب ساده‌ی

$p = q$ (قسمت ۴ در ص ۴۴) عملیات زیر باید انجام شود:

الف) جایی که q اشاره می‌کند، RC آن را یک واحد کم می‌کنیم.

ب) اگر $RC = 0$ آن خانه را به فضای در دسترس (استخر طوفانی) برگردانیم.

ج) مقدار p را به q نسبت بده.

د) به شمارنده‌ی ارجاع (RC) جایی که p اشاره می‌کند، یک واحد اضافه کن.

همان‌طور که ملاحظه می‌شود، این دستور انتساب ساده هزینه‌ی زیادی دارد؛ چراکه باید

شمارنده‌های ارجاع را اصلاح کند. علاوه بر این فضای اضافی برای نگهداری تعداد ارجاع

نیاز است. به همین دلیل، این روش در سیستم‌های پردازش موازی، متداول است.

نکته * بر روی دو مشکل اشاره گر سرگردان و متغیر پویای هری گم شده، مشکل اول فیلد

عادت تر است.

۲- تنبل (lazy): زباله روبی: از مشکلات اشاره گر معلق و زباله، به تدریج سری اشاره گر

معلق مشکل مهم تری است؛ چرا که به دلیل تغییر تهاوی حافظه‌ی در حال استفاده، منجر به

هرج و مرج می شود. فلسفه‌ی زباله روبی این است که اجازه داده می شود تا زباله تولید شود،

تا از اشاره گر معلق، پرهیز شود. برای این منظور دستور delete در اختیار ما قرار داده

نمی شود. مثلاً زبان های C# و Java اینگونه اند. اگر اسیب فضای آزاد (استغنی حافظه)

کاملاً فانی شده باشد و نیاز به حافظه باشد، روال زباله روبی اجرامی شود تا عناصر زباله‌ی

موجود در heap را شناسایی و آن را به اسیب فضای آزاد برگرداند. برای این کار دو مرحله‌ی

زیر انجام می شود:

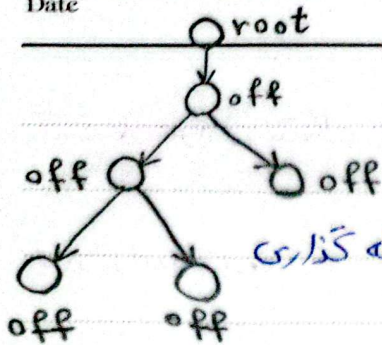
الف) مرحله‌ی نشانه گذاری: در این مرحله هر عنصر فعال موجود در heap که بخشی از حافظه‌ی

داره‌ی قابل دسترسی می باشد، نشانه گذاری می شود. هر عنصر باید شامل بیت زباله روبی

باشد که ابتدا 0n (۱) است. الگو ریتیم نشانه گذاری، بیت زباله روبی هر عنصر فعال

بیت زباله رو بن بررسی می شود. اگر بیت off باشد، از آن عبور می کند و اگر on باشد، آن
 مدخل به لیست فضای آزاد برگردانده (۴۸) می شود.

Subject:
 Date:



را off (مدخل) می کند.

ب) بار کردن: وقتی الگوریتم نشانه گذاری، عناصر فعال را نشانه گذاری

کرد، تمامی عناصری که بیت زباله رو بن آن ها on است به عنوان زباله محسوب می شوند

و به لیست فضای آزاد بازگردانده می شوند. برای این کار جستجوی ترتیبی heap انجام می شود.

پروسیجر

زیر برنامه ها (تابع، function، procedure):
 {
 int x, out int y: پارامترهای رسمی
 {
 int z; (formal parameters)
 return z;
 }
 }
 (مثال در C#)

main()
 {
 int a و b و (actual parameters)
 a = 10 و پارامترهای واقعی
 cout << fun(a و b) و
 }
 روشن های ارسال پارامتر

۱- ارسال با مقدار (by value): در این روش، مقدار پارامتر واقعی، به مقداردهی

اولیه به پارامتر رسمی متناظر استفاده می شود. پس از این مقدار اولیه که در هنگام فراخوانی

انجام می شود، از پارامتر رسمی می توان به صورت یک متغیر محلی در زیر برنامه، استفاده نمود.

ارسال با مقدار با استفاده از کپی مقدار ، انجام می شود . در نتیجه برای پارامترهای رسمی نیاز به فضای ذخیره سازی داریم که مثلاً در ارسال یک آرایه ی بزرگ ، می تواند خیلی پرهزینه باشد . هالش ورودی (in) است . (پارامتر a در مثال قبل ، از این نوع است .)

۲- ارسال با نتیجه (by result) : هالش خروجی است . هنگامی که یک پارامتر به این صورت ارسال می شود ، هیچ مقداری به زیر برنامه انتقال نمیابد . (یعنی تواند مقدار باشد .)

از پارامتر رسمی به عنوان یک متغیر محلی استفاده می شود و بعد از اینکه کنترل به فراخواننده برمی گردد ، مقدار پارامتر رسمی ، به پارامتر واقعی کپی می شود . و امن است که پارامتر واقعی با یک متغیر باشد . (پارامتر a در مثال قبل ، از این نوع است .)

۳- ارسال با مقدار - نتیجه (pass by value - result) :
یا call

حالت ورودی - خروجی (in - out) دارد . به آن ارسال با کپی نیز می گویند ؛ چرا که ۲

تا کپی انجام می شود . در زمان فراخوانی ، مقدار پارامتر واقعی در رسمی کپی می شود و با

خانه ی زیر برنامه مقدار پارامتر رسمی در واقعی کپی می شود .

اگر در مثال قبل ، fun را به این صورت تغییر
دهیم ، تا قبل از پایان fun ، a هنوز ۱۰ است .
پایان fun ، a داخل main ، ۱۱ می شود . $a = 10$ ، $x = 11$ → و $x++$

۴- ارسال با ارجاع (by reference):

```
int fun(int& x, int& y)
```

```
{
```

```
    x++
```

```
    y += 10
```

```
}
```

```
main()
```

```
{
```

```
    int a و b
```

```
    a = 10 و b = 20
```

```
    cout << fun(a و b)
```

```
}
```

در این روش، آدرس متغیر در حافظه منتقل می‌شود.

آدرس متغیر x: 10
آدرس متغیر y: 20

مزیت بزرگ آن کارایی است؛ چرا که نام متغیر

(آلیاس)

از نظر زمان و مکان، کارایی بالاتر دارد؛ چرا که در این روش

هیچ زمانی به کپی کردن مقادیر، صرف نمی‌شود و

نیاز به هیچ گونه فضای اضافی نیست. عیب آن نیز

همان عیب نام مستعار (آلیاس) است. مستعارسازی یا استفاده از نام مستعار، توانایی

برنامه را کاهش می‌دهد. در نتیجه قابلیت اعتماد کم می‌شود. مورد آن in-out است.

۵- ارسال با نام (by name): این روش در زبان الگول % استفاده شده است.

مورد آن ورودی - خروجی (in-out) است. در این روش ارسال، پارامترهای واقعی به

طرح متنی، جایگزین پارامترهای رسمی متناظر آن‌ها در تمام قسمت‌های برنامه می‌شوند.

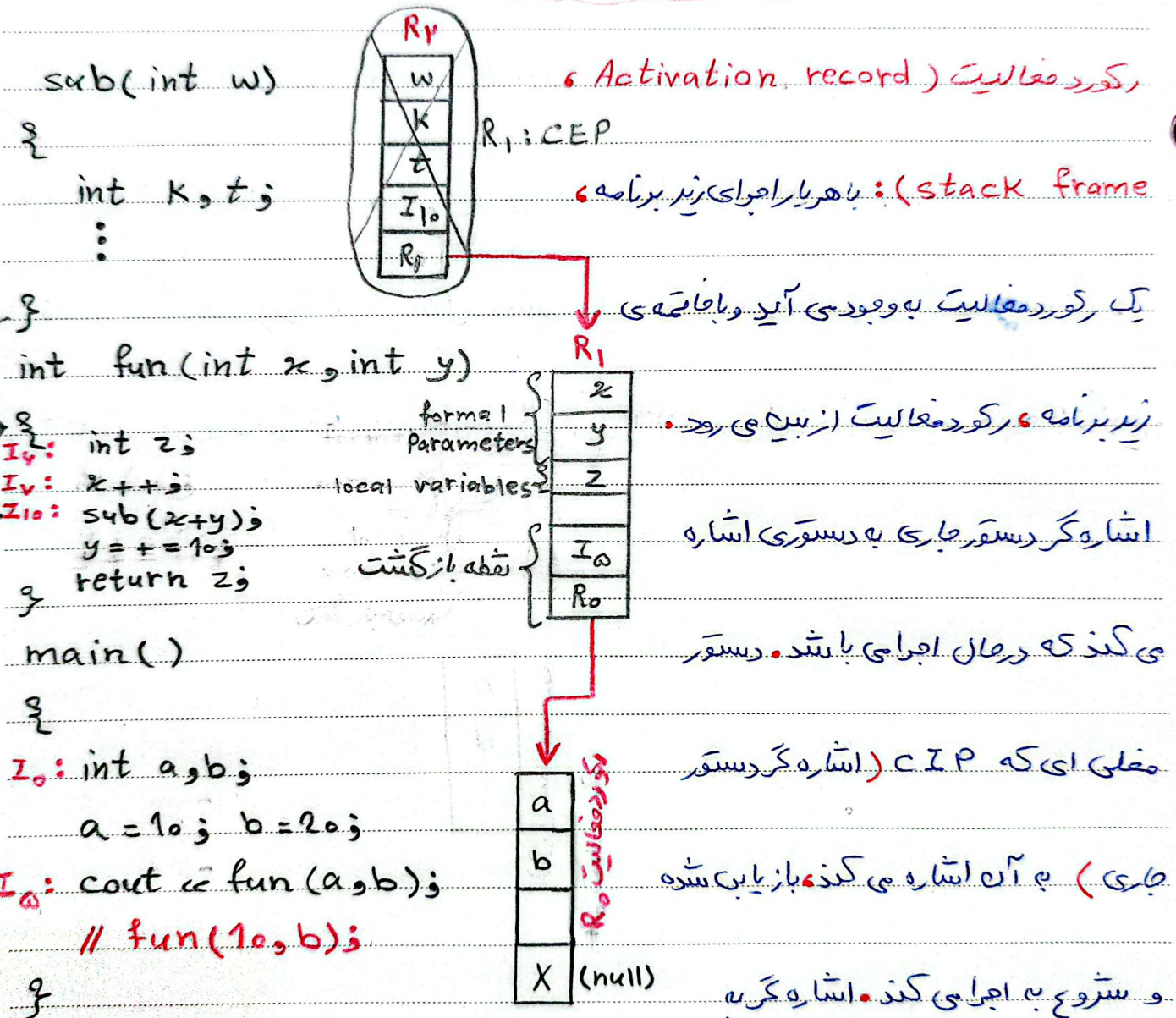
در نتیجه کاملاً متفاوت با روش‌های قبلی است؛ چرا که در هنگام فراخوانی، مقیدسازی انجام

نمی‌شود. بلکه به تأخیر می‌اندد تا از پارامتر رسمی استفاده شود. (مقیدسازی یعنی همان وقتی که

$z = \text{stack dynamic variable}$ نسبت دادن مقدار پارامتر واقعی به رسمی

سه حالت پارامتر: ورودی (in)، خروجی (out)، ورودی-خروجی (in-out).

پایه سازی زیر برنامه ها



مغلی جاری ، اشاره گر به مغلی فعالیت مورد اشاره است. وقت کنیز که ممکن است چندین CEP

P4PCO CEP: R₀ R₁ R_p CIP: I₁₀ I₄ I₃ I₂ I₁

Subject:

Date

(۵۲)

رکورد فعالیت، حتی برای یک زیر برنامه وجود داشته باشد. CEP به رکورد فعالیت مغلی اشاره می کند. با فراخوانی زیر برنامه و ایجاد رکورد فعالیت برای آن، CEP به آن رکورد فعالیت و CIP به اولین دستور آن در کد سلامت، اشاره خواهد کرد. قبل از دادن این مقادیر جدید، مقادیر قبلی CIP و CEP باید ذخیره شوند. آنها را در رکورد فعالیت زیر برنامه ای که فراخوانی می شود، ذخیره می کند. در شیء داده ای به نام «نقطه ی بازگشت» نقطه ی بازگشت (برگشت) شامل فضایی برای ذخیره سازی ۲ مقدار اشاره گر است:

CIP، دستوری که می خواهیم به آن بازگردیم و CEP، محیطی که می خواهیم به آن بازگردیم.