به نام خدا

زبان‌های برنامه‌نویسی

آرش شفیعی

برنامه نویسی رویه‌ای

برنامه نویسی رویه‌ای

- برنامه نویسی رویه‌ای [1] بر اساس مفهوم فراخوانی رویه [2] است. یک برنامه در یک زبان برنامه نویسی رویه‌ای تشکیل شده است از تعدادی رویه که یکدیگر را فراخوانی می‌کنند. رویه‌ها بر خلاف توابع در برنامه نویسی تابعی دارای حالت هستند بدین معنی که خروجی یک رویه می‌تواند بسته به حالت سیستم به ازای یک ورودی یکسان تغییر کند. یک رویه تشکیل شده است از تعدادی انتساب متغیر که مقدار آنها می‌تواند درون رویه یا بیرون رویه تغییر کند و تعداد ساختار کنترلی [3] و حلقه [4] که در مورد آنها صحبت خواهیم کرد.
- بنابراین محاسبات در برنامه نویسی رویه‌ای توسط ارزیابی عبارت و انتساب مقادیر به متغیرها حاصل می‌شود. علاوه بر عبارات با استفاده از ساختارهای کنترلی می‌توان از بین چند مسیر کنترلی برای محاسبات یک مسیر را انتخاب کرد و با استفاده از حلقه‌ها اجرای دسته‌ای از عبارات محاسباتی را تکرار کرد.

---

[1] procedural programming
[2] procedure call
[3] control structure
[4] loop

ساختارهای کنترلی

- ساختارهای کنترلی برای اولین بار در زبان فورترن به وجود آمدند که برای معماری ماشین آی‌بی‌ام ۷۰۴ به وجود آمد. زبان این ماشین نیز ساختارهای کنترلی را شامل می‌شد که در زبان فورترن دستور `goto` نام داشت.
- در دهۀ ۱۹۶۰ اثبات شد که همۀ برنامه‌ها تنها توسط دو ساختار کنترلی یکی برای انتخاب بین چند مسیر پردازش و دیگری برای تکرار دسته‌ای از محاسبات می‌توانند ساخته شوند و این دو ساختار نیاز همه برنامه‌ها را برآورده می‌کند، علاوه بر اینکه دستور `goto` که برای ارجاع کنترل برنامه به یکی از دستورات برنامه طراحی شده بود می‌توانست خطرساز باشد. دلیل این خطرسازی این بود که برنامه نویس می‌توانست کنترل برنامه را به هر جایی که می‌خواست ارجاع دهد و این باعث پیچیدگی برنامه می‌شد و می‌توانست احتمال بروز خطا در منطق برنامه را افزایش دهد. علاوه بر این، با استفاده از ساختارهای کنترلی جدید، خوانایی برنامه‌ها بیشتر می‌شد.

برنامه نویسی رویه‌ای

- یک ساختار کنترلی انتخاب می‌کند که کدام یک از مجموعه دستورات برای اجرا انتخاب شوند.
- یک عبارت انتخاب [1] وسیله‌ای است برای انتخاب یکی از مسیرها در اجرای یک برنامه.
- در بیشتر زبان‌ها انتخاب کننده دو حالته توسط `if-then-else` و انتخاب کننده چند حالته توسط `switch-case` پیاده سازی شده است.
- یک عبارت تکرار [2] وسیله‌ای است برای تکرار مجموعه‌ای از عبارت به تعداد صفر، یک یا چند بار.
- ساختار تکرار معمولاً حلقه نامیده می‌شود که در بیشتر برنامه‌ها با `for` و `while` و `foreach` پیاده سازی شده است.

[1] selection statement
[2] iterative statement

برنامه نویسی رویه‌ای

- زبان پایتون یک سازوکار جدید برای استفاده در ساختارهای تکرار مهیا کرده است.
- توابع معمولی در پایتون و دیگر زبان‌های برنامه نویسی یک مقدار را محاسبه کرده و بازمی‌گردانند. حال فرض کنید در یک حلقه نیاز داریم تابعی را فراخوانی کنیم که به ازای هر بار فراخوانی یک مقدار جدید را محاسبه کرده، بازمی‌گرداند. بدین ترتیب دنباله‌ای از مقادیر محاسبه شده به ترتیب از تابع بازگردانده می‌شوند.
- چنین ساز و کاری توسط مولدها [1] در پایتون قابل پیاده سازی است.
- در توابع معمولی بعد ازینکه مقداری توسط یک تابع بازگردانده می‌شود، متغیرهای محلی از بین می‌روند و در فراخوانی بعدی تابع، متغیرها دوباره ساخته و مقدار دهی می‌شوند. اما در مولدها پس از اینکه مقداری از یک مولد بازگردانده شد، متغیرهای محلی از بین نمی‌روند و در فراخوانی بعدی، مولد به عملیات ادامه می‌دهد.

[1] generators

برنامه نویسی رویه‌ای

- مولد زیر را در نظر بگیرید :

```
۱ def generate_ints(N) :
۲     for i in range(N) :
۳         yield i
```

- هر تابعی که از کلمه yield استفاده کند، یک مولد است. با بازگرداندن یک مقدار توسط کلمه yield اجرای تابع به اتمام نمی‌رسد بلکه متوقف می‌شود و در فراخوانی بعدی ادامه پیدا می‌کند. فراخوانی‌های بعدی توسط تابع next() انجام می‌شوند. تابع مولد در واقع یک پیمایشگر بازمی‌گرداند.

```
۱ gen = generate_ints (3)
۲ next (gen)   # 0
۳ next (gen)   # 1
۴ for i in generate_ints (5) :
۵     print (i)
```

برنامه نویسی رویه‌ای

- به عنوان مثال دیگر فرض کنید مولدی می‌خواهیم که رئوس یک گراف را پیمایش کند و در هر بار فراخوانی، رأس پیمایش شدهٔ بعدی را بازگرداند.
- می‌توانیم مولدی به صورت زیر بنویسیم :

```
۱ def traverse (G) :
۲     # if there is another node :
۳     #       set nd to the next node
۴     # else :
۵     #       return
۶     yield nd
```

- در هر بار فراخوانی توسط تابع `next()` تابع `traverse` یک رأس پیمایش شده را بازمی‌گرداند.

زیر برنامه‌ها

- زیر برنامه‌ها [1] که رویه [2] یا تابع [3] نیز نامیده می‌شوند، به برنامه نویسان کمک می‌کنند تا مجموعه‌ای از محاسبات را جدا کرده و با نامی انتزاعی تعریف کنند تا این دسته از محاسبات بتوانند دوباره مورد استفاده قرار بگیرند. علاوه بر این که زیر برنامه‌ها باعث صرفه جویی در زمان برنامه نویسی و مصرف حافظه در برنامه می‌شوند، خوانایی برنامه را نیز بهبود می‌دهند.

- ویژگی‌های همهٔ زیر برنامه‌ها (به غیر از زیر برنامه‌های موازی که در برنامه نویسی همروند استفاده می‌شوند) به شرح زیر است :

  ۱. هر زیر برنامه در یک نقطه مقداردهی اجرای (فراخوانی) آن آغاز می‌شود.
  ۲. اجرای برنامه‌ای که یک زیر برنامه را فراخوانی می‌کند متوقف می‌شود تا زیر برنامه محاسبات خود را انجام دهد، بنابراین در هر لحظه فقط یک زیر برنامه در حال اجراست.
  ۳. پس از اتمام اجرای زیربرنامه، کنترل اجرای محاسبات به برنامه‌ای که زیربرنامه را فراخوانی کرده است بازگردانده می‌شود.

---

[1] subprograms
[2] procedure
[3] function

زیر برنامه‌ها

- تعریف زیر برنامه [1] توصیف می‌کند چه عملیاتی توسط یک زیر برنامه انجام می‌شود. در تعریف یک زیر برنامه یک نام انتزاعی برای زیر برنامه تعیین می‌شود و تعدادی متغیر ورودی برای زیر برنامه تعریف می‌شوند. در زبان‌هایی که نوع متغیرها در زمان کامپایل مشخص می‌شوند، نوع ورودی‌ها و خروجی‌ها نیز مشخص می‌شوند.
- فراخوانی زیر برنامه [2] درخواستی است که توسط یک برنامه یا یک زیر برنامهٔ دیگر برای اجرای یک زیر برنامه صادر می‌شود.
- نام یک زیر برنامه به همراه ورودی و خروجی‌های آن اعلام زیر برنامه [3] برنامه نامیده می‌شود.

[1] subprogram definition
[2] subprogram call
[3] declaration

زیر برنامه‌ها

- در برخی زبان‌ها تعریف تابع از اعلام آن جدا می‌شود و اعلام زیر برنامه در یک فایل جداگانه قرار می‌گیرد. دو دلیل برای این کار وجود دارد، یکی اینکه برنامه نظم بیشتری پیدا می‌کند و به اعلام زیر برنامه‌ها (به عنوان مستندات راهنما) راحت‌تر می‌توان دسترسی پیدا کرد. دلیل دوم این است که گاهی یک برنامه تجاری است و تعریف زیر برنامه‌ها نمی‌تواند در اختیار کاربران قرار بگیرد ولی اعلام آن جهت استفاده باید در دسترس باشد.
- در زبان سی و سی++ به اعلام زیر برنامه‌ها پروتوتایپ گفته می‌شود.

زیر برنامه‌ها

- برای اینکه زیر برنامه‌ها بتوانند به متغیرهایی که غیر محلی هستند (توسط خود زیر برنامه تعریف نشده است) دسترسی پیدا کنند، دو روش وجود دارد. در روش اول زیر برنامه به متغیر به طور مستقیم دسترسی دارد، بدین معنی که متغیری به طور عمومی تعریف شده و زیر برنامه به آن به طور مستقیم دسترسی دارد. در روش دوم متغیرهای مورد نیاز زیر برنامه به عنوان ورودی به زیر برنامه فرستاده می‌شوند.
- استفاده از متغیرهای عمومی خوانایی برنامه و همچنین قابلیت اطمینان برنامه را پایین می‌آورند زیرا دنبال کردن و تست برنامه سخت‌تر می‌شود چرا که زیر برنامه فقط به ورودی‌هایش بستگی ندارد بلکه ممکن است زیر برنامه‌های دیگر یک متغیر عمومی را تغییر دهند و نتیجه یک زیر برنامه تغییر کند. در زبان‌های تابعی امکان تعریف متغیر وجود ندارد که باعث افزایش قابلیت اطمینان و ساده‌تر شدن برنامه‌ها برای تحلیل و بررسی می‌شود.

زیر برنامه‌ها

- در برخی از زبان‌ها می‌توان در یک زیر برنامه به عنوان ورودی به جای داده، یک زیربرنامه دریافت کرد. بنابراین در چنین حالتی ورودی زیربرنامه یک متغیر است که می‌تواند به نام یکی از زیربرنامه‌ها ارجاع داده شود.
- ورودی زیر برنامه‌ها در تعریف زیر برنامه پارامتر [1] نیز نامیده می‌شوند. زمان انقیاد حافظهٔ پارامترهای یک زیر برنامه در هنگام فراخوانی زیر برنامه است.
- در زمان فراخوانی یک زیر برنامه نام زیر برنامه به همراه ورودی‌های آن مشخص می‌شوند. ورودی یک برنامه در هنگام فراخوانی آرگومان [2] های زیربرنامه نامیده می‌شوند.

[1] parameter
[2] argument

زیر برنامه‌ها

- در بیشتر زبان‌های برنامه نویسی تناظر آرگومان‌ها و پارامترها یا به عبارت دیگر انقیاد مقدار پارامترها با استفاده از مقدار آرگومان‌ها توسط موقعیت آنها در فراخوانی انجام می‌شود، بدین معنی که مقدار اولین پارامتر به مقدار اولین آرگومان مقید می‌شود و بدین ترتیب الی آخر.
- وقتی تعداد پارامترها زیاد می‌شود، ممکن است این نوع فراخوانی باعث ایجاد خطا در برنامه نویسی شود.

زیر برنامه‌ها

- در برخی زبان‌ها مانند پایتون می‌توان آرگومان‌ها را به پارامترها منسوب‌کرد و بدین صورت ترتیب آرگومان‌ها در هنگام فراخوانی بی‌اهمیت می‌شود.
- برای مثال :

```
۱ def logger (time, message) :
۲       print ("[" + time + "]" + message)
۳ logger (message = "error" , time = "7am")
```

زیر برنامه‌ها

- همچنین می‌توان برای یک پارامتر مقدار پیش‌فرض [1] نیز تعیین کرد.

```
def compute_pay (incom , exemption = 1 , tax_rate) :
        ...
pay = compute_pay (2000 , tax_rate = 0.15)
```

- توجه کنید که در اینجا آرگومان اول به پارامتر اول مقید می‌شود و آرگومان دوم به پارامتر سوم که نام پارامتر برای آن مشخص شده است.
- در زبان سی++ که تعیین پارامتر در زمان فراخوانی امکان پذیر نیست، پارامترها با مقادیر پیش فرض باید در پایان لیست پارامترها تعریف شوند.

[1] default

زیر برنامه‌ها

- در یک تقسیم بندی به زیر برنامه‌هایی که مقداری را بازنمی‌گردانند رویه [1] و به زیر برنامه‌هایی که مقدار بازمی‌گردانند تابع [2] گفته می‌شود ولی در بیشتر زبان‌ها همه زیر برنامه‌ها مقدار نیز بازمی‌گردانند.
- تنها زبان‌های قدیمی مانند فورترن و آدا رویه بدین معنا دارد.

[1] procedure
[2] function

زیر برنامه‌ها

- یک زیر برنامه عمومی [1] زیر برنامه‌ای است که در آن پارامترهای می‌توانند نوع عمومی داشته باشند.
- در زبان‌هایی که نوع متغیرها در زمان اجرا تعیین می‌شوند، یک زیر برنامه می‌تواند انواع متفاوتی از ورودی‌ها را بپذیرد و به عبارتی زیر برنامه به صورت عمومی تعریف می‌شود.
- برای مثال تابع زیر می‌تواند رشته یا عدد دریافت کند.

```
۱ def add (a, b) : return a+b
۲ add(2,3) # 5
۳ add(2.3, 4.5) # 6.8
۴ add("hello, ", "world") # "hello, world"
```

[1] generic subprogram

زیر برنامه‌ها

- در زبان‌هایی که نوع متغیرها در زمان کامپایل تعیین می‌شود، نوع متغیر ورودی تابع نمی‌تواند تغییر کند، مگر اینکه سازوکار برنامه‌نویسی عمومی در آن زبان وجود داشته باشد.
- قالب‌ها در زبان سی++ [1] برای طراحی توابع با نوع داده‌ای عمومی تعریف شده‌اند. برای مثال با استفاده از برنامه‌نویسی عمومی تابع زیر می‌تواند رشته یا عدد دریافت کند.

```
۱  template <class T>
۲  T add(T a, T b) {
۳      return a+b;
۴  }
۵  int main() {
۶      int x = add<int>(2,3);
۷      float y = add<float>(2.3, 4.5);
۸      string s = add<string>("hello ", "world");
۹      cout << x << y << s << endl;
۱۰ }
```

[1] template

زیر برنامه‌ها

- در برخی از زبان‌ها مانند پایتون یک زیر برنامه می‌تواند در یک زیر برنامه تعریف شود، اما در برخی زبان‌ها مانند سی این کار امکان پذیر نیست.
- برای مثال تابع private تنها در تابع f قابل دسترس است.

```
۱ def f(...) :
۲     ...
۳     def private(...) :
۴         ...
۵     private(...)
۶     ...
```

- در زبان سی در یک تابع نمی‌توان تابع تعریف کرد و در نتیجه توابع تعریف شده توسط همهٔ توابع دیگر در دسترس هستند.

زیر برنامه‌ها

- همچنین برخی از زبان‌ها مانند پایتون اجازه می‌دهند یک تابع به عنوان آرگومان به تابع دیگر ارسال شود، اما در برخی زبان‌ها مانند سی این کار تنها توسط اشاره‌گر به تابع امکان پذیر است.
- برای مثال فرض کنید می‌خواهیم برنامه‌ای بنویسیم که یک تابع را دریافت کند و انتگرال آن را در بازهٔ a و b محاسبه کند. در پایتون می‌توانیم این برنامه را به صورت زیر بنویسیم.

```
۱ def integral(f, a, b, d=0.001) :
۲   return reduce(add, (map(lambda x : f(x)*d, np.arange(a,b,d))))
۳
۴ cube = lambda x : x*x*x
۵
۶ integral(cube, 0, 10)
۷ # 2499.5000250000016
```

زیر برنامه‌ها

- در زبان سی برای ارسال تابع به تابع باید از اشاره‌گر به تابع [1] استفاده کنیم. برای این کار باید از امضای تابع [2] استفاده کنیم. امضای تابع مشخص می‌کند یک تابع چند ورودی از چه نوع‌های داده دارد و نوع دادهٔ خروجی آن چیست.
- برای مثال یک اشاره‌گر به تابع با دو ورودی عدد صحیح و اعشاری و یک خروجی منطقی را به صورت زیر تعریف می‌کنیم. سپس این اشاره‌گر به تابع را می‌توانیم با نام یک تابع مقداردهی کنیم. پس نام توابع در واقع اشاره‌گر به تابع هستند.

```
۱ bool function(int i, double d) {
۲     // ...
۳ }
۴
۵ bool (*ptr) (int, double);
۶ ptr = function;
```

[1] function pointer
[2] function signature

زیر برنامه‌ها

- فرض کنید می‌خواهیم به چند تابع توسط آرایه‌ای از اشاره‌گرها به توابع دسترسی پیدا کنیم.

```
۱  double add(double x, double y) { return x+y; }
۲  double sub(double x, double y) { return x-y; }
۳  double mul(double x, double y) { return x*y; }
۴  double div(double x, double y) { return x/y; }
۵
۶  int main() {
۷      double (*op[])(double, double) = { add, sub, mul, div };
۸      int i;
۹      double a,b;
۱۰     scanf("%d", i);
۱۱     scanf("%f", a); scanf("%f", b);
۱۲     if (i>=0 && i<4)
۱۳         op[i](a, b);
۱۴     return 0;
۱۵ }
```

زیر برنامه‌ها

- همچنین می‌توانیم تابعی تعریف کنیم که در ورودی یک تابع را دریافت می‌کند. برای این کار از اشاره‌گر به تابع استفاده می‌کنیم.

```
۱  double operation(double x, double y, double(*op)(double, double)) {
۲      return op(x,y);
۳  }
۴
۵  int main() {
۶      double res;
۷      res = operation(8, 3, div);
۸
۹      return 0;
۱۰ }
```

زیر برنامه‌ها

- در برخی زبان‌ها امکان سربارگذاری زیر برنامه‌ها [1] وجود دارد، بدین معنی که چند زیر برنامه با نام یکسان می‌توانند پارامترها با نوع‌های متفاوت دریافت کنند.
- در برنامه زیر در فراخوانی اول تابع add تابع اول و در فراخوانی دوم تابع دوم اجرا می‌شود.

```
۱  void add(int a, int b) {
۲          cout << a+b;
۳  }
۴  void add(string a, string b) {
۵          cout << a+b;
۶  }
۷  int main() {
۸          add(5, 3);
۹          add("hello ", "world");
۱۰ }
```

[1] overload subprogram

زیر برنامه‌ها

- متغیرها در زیر برنامه‌ها محلی هستند بدین معنی که تنها در هنگام فراخوانی زیر برنامه به حافظه مقید می‌شوند. گاهی نیاز به متغیرهایی است که در فراخوانی‌های متفاوت یک زیر برنامه مقدار خود را از دست نمی‌دهند و در حافظه باقی می‌مانند. در زبان سی و سی++ تعریف چنین متغیرهایی توسط واژهٔ `static` امکان پذیر است. بدین ترتیب متغیر در حافظه پشته [1] متناظر با زیر برنامه ساخته نمی‌شود بلکه در قسمت داده‌های برنامه [2] تعریف می‌شود.

[1] stack memory
[2] data segment

روش‌های ارسال پارامتر

- روش‌های متعددی برای ارسال پارامتر به یک تابع زیر برنامه وجود دارد که در اینجا به آنها اشاره می‌کنیم.
- پارامترها می‌توانند سه مدل معنایی متفاوت داشته باشند. به عبارت دیگر پارامترها به سه روش متفاوت با آرگومان‌ها رفتار می‌کنند. در روش اول پارامترها تنها آرگومان‌ها را دریافت می‌کنند. چنین پارامترهایی تنها دارای حالت ورودی [1] هستند. در روش دوم پارامترها به آرگومان‌ها مقدار ارسال می‌کنند. این پارامترها تنها دارای حالت خروجی [2] هستند. در روش سوم، پارامترها هم از آرگومان‌ها مقدار دریافت می‌کنند و هم به آنها مقدار ارسال می‌کنند. این پارامترها دارای حالت ورودی و خروجی [3] هستند.

[1] in mod
[2] out mod
[3] in out mod

روش‌های ارسال پارامتر

- وقتی یک آرگومان با مقدار ارسال شده [1] یا به عبارت دیگر یک پارامتر با مقدار دریافت شده است، مقدار آرگومان به عنوان مقدار اولیه برای پارامتر در نظر گرفته می‌شود، اما آرگومان و پارامتر به دو مکان متفاوت در حافظه اشاره می‌کنند، بنابراین مقدار آرگومان در فضای مربوط به پارامتر کپی می‌شود.
- عیب ارسال با مقدار این است که ممکن است حجم یک متغیر زیاد باشد و کپی کردن آن سرعت اجرای برنامه را کاهش می‌دهد.
- متغیرهای اصلی زبان سی با مقدار ارسال می‌شوند.

[1] pass by value

روش‌های ارسال پارامتر

- وقتی یک آرگومان با نتیجه ارسال می‌شود [1] ، هیچ مقداری به زیر برنامه فرستاده نمی‌شود بلکه زیر برنامه مقدار متغیر را تغییر می‌دهد و نتیجه را به برنامه فراخوانی کنندهٔ زیر برنامه می‌فرستد. به عبارت دیگر پس از محاسبهٔ یک پارامتر مقدار آن در مقدار آرگومان کپی می‌شود.
- یکی از معایب ارسال با نتیجه این است که در مقدار دهی‌ها ممکن است تصادم [2] اتفاق بیافتد. فرض کنید یک متغیر دو بار به عنوان دو آرگومان در یک فراخوانی به یک زیر برنامه ارسال می‌شود. برای مثال `sub(p1,p1)` . حال در این فراخوانی `p1` دو بار مقدار دهی می‌شود ولی در نهایت مقداری را به خود می‌گیرد که برای بار آخر به آن داده شده است.

[1] pass by result
[2] collision

روش‌های ارسال پارامتر

- برای مثال تابع زیر در زبان سی شارپ را در نظر بگیرید. دو متغیر ورودی در این تابع ارسال با نتیجه می‌شوند.

```
۱ void Set(out int x , out int y ) {
۲       x = 17;
۳       y = 35;
۴ }
۵ Set(out a , out a) ;
```

- مقدار a در نهایت برابر با ۳۵ می‌شود، گرچه ابتدا مقدار آن برابر با ۱۷ قرار گرفته است.
- به علت مشکلاتی که همه ارسال با نتیجه ایجاد می‌کند، بسیاری از زبان‌ها این روش را پشتیبانی نمی‌کنند.

روش‌های ارسال پارامتر

- ارسال با مقدار و نتیجه [1] ترکیبی از ارسال با مقدار و ارسال با نتیجه است.
- مقدار آرگومان ابتدا در پارامتر کپی می‌شود و پارامتر مقدار اولیه خود را می‌گیرد و در نهایت مقدار پارامتر در مقدار آرگومان کپی می‌شود.

[1] pass by value result

روش‌های ارسال پارامتر

- ارسال با ارجاع [1] روش دیگری برای پیاده سازی حالت ورودی و خروجی است.
- در این روش به جای کپی کردن مقدار آرگومان در پارامتر و سپس کپی کردن مقدار پارامتر در آرگومان که دارای سربار کپی است، مکان حافظه متغیر آرگومان به عنوان پارامتر به زیر برنامه ارسال می‌شود و زیر برنامه می‌تواند آن مکان حافظه را تغییر دهد.
- ارسال با ارجاع در سی و سی++ با استفاده از اشاره‌گرها انجام می‌شود و در سی++ با متغیر ارجاعی نیز این کار امکان پذیر است.

[1] pass by reference

روش‌های ارسال پارامتر

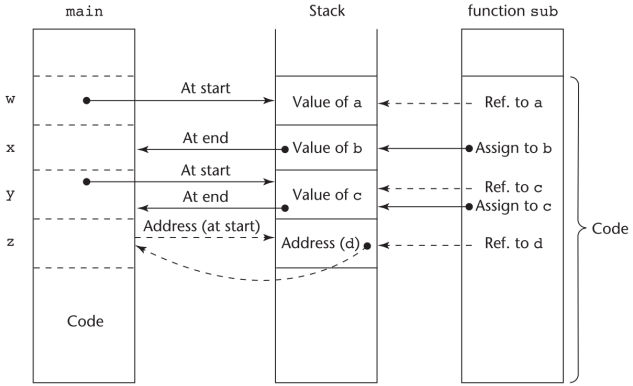
- ارسال با ارجاع چند مزیت دارد. مزیت اول صرفه جویی در زمان اجرا است چرا که در ارسال با ارجاع برای متغیرهای با حجم زیاد در زمان کپی صرفه جویی می‌شود و مزیت دوم صرفه‌جویی در حافظه است چرا که در زمان اجرا نیاز به تخصیص حافظه‌های اضافی وجود ندارد.
- یکی از معایب ارسال با ارجاع این است که ممکن است بخواهیم برای صرفه‌جویی در زمان از ارسال با ارجاع استفاده کنیم، اما نخواهیم زیر برنامه مقدار پارامتر را تغییر دهد. در زبان سی و سی++ برای این کار از اشاره‌گر ثابت استفاده می‌شود و در زبان سی++ با استفاده از متغیر ارجاعی ثابت نیز امکان پذیر است.

روش‌های ارسال پارامتر

- در ارسال پارامتر با مقدار در واقع مقدار آرگومان در پشته مربوط به زیر برنامه کپی می‌شود و زیر برنامه بر روی پشته به مقدار آرگومان دسترسی دارد.
- در ارسال پارامتر با ارجاع آدرس آرگومان در پشته مربوط به زیر برنامه کپی می‌شود و با اعمال تغییرات توسط زیر برنامه در آن مکان حافظه، مقدار متغیر نیز تغییر می‌کند که توسط برنامه فراخوانی کننده قابل مشاهده است.
- در ارسال پارامتر با نتیجه مقدار متغیر توسط زیربرنامه در پشته کپی می‌شود و برنامه مقدار مورد نظر را از پشته دریافت و در آرگومان کپی می‌کند.
- در ارسال پارامتر با مقدار و نتیجه برنامه مقدار آرگومان را در پشته کپی می‌کند، زیر برنامه برنامه محاسبات مورد نیاز را انجام می‌دهد و در نهایت نتیجه را در پشته کپی می‌کند و برنامه فراخوانی کننده نتیجه را در آرگومان کپی می‌کند.

روش‌های ارسال پارامتر

- روش پیاده سازی انواع ارسال پارامتر در شکل زیر نشان داده شده است.



Function header: **void** sub (**int** a, **int** b, **int** c, **int** d)
Function call in main: sub (w,x,y,z)
(pass w by value, x by result, y by value-result, z by reference)

روش‌های ارسال پارامتر

- در زبان سی ارسال با ارجاع توسط اشاره‌گرها پیاده سازی شده است که قبل از آن در زبان الگول نیز وجود داشت.
- اگر یک پارامتر یک اشاره‌گر ثابت باشد، فراخوانی سریع‌تر است چرا که در زمان کپی مقدار آرگومان در مقدار پارامتر صرفه جویی می‌شود.
- در زبان سی++ متغیر ارجاعی علاوه بر اشاره‌گر وجود دارد که در واقع یک نام مستعار برای یک مکان در حافظه است. پس از مقداردهی اولیه یک متغیر ارجاعی امکان مقدار دهی مجدد وجود ندارد بدین معنی که یک متغیر ارجاعی همیشه فقط به یک مکان حافظه اشاره می‌کند.
- در زبان جاوا متغیرها از نوع اصلی با مقدار و اشیاء از نوع کلاس‌ها با ارجاع ارسال می‌شوند، چرا که اشیاء همگی در واقع متغیر ارجاعی هستند.

روش‌های ارسال پارامتر

- در زبان پایتون برخی از متغیرها قابل تغییر [1] هستند و برخی دیگر غیر قابل تغییر [2] اند.
- لیست‌ها و دیکشنری‌ها و مجموعه‌ها قابل تغییراند ولی رشته‌ها و چندتایی‌ها غیر قابل تغییراند. همچنین نوع‌های عددی مانند اعداد صحیح و اعشاری غیر قابل تغییراند.

[1] mutable
[2] immutable

روش‌های ارسال پارامتر

- اگر یک متغیر قابل تغییر به یک تابع ارسال شود، ارسال با ارجاع است، بنابراین تابع می‌تواند مقدار آن را تغییر دهد.
- اگر تابع متغیر قابل تغییر را به مقدار جدید مقید کند یا به عبارت دیگر انقیاد مجدد [1] صورت بگیرد، و در نتیجه مکان حافظه تغییر کند، برنامه فراخوانی کننده از این انقیاد بی اطلاع خواهد بود. به عبارت دیگر، زیر برنامه‌ها می‌توانند متغیرهای قابل تغییر را تغییر دهند ولی مکان حافظه آنها را نمی‌توانند تغییر دهند.
- اگر یک متغیر غیر قابل تغییر به یک تابع ارسال شود تابع نمی‌تواند مقدار آن را تغییر دهد.

[1] rebind

روش‌های ارسال پارامتر

- برای مثال :

```
۱ def change (lst) :
۲      lst.append ('x')  #  value of list changes
۳      lst = ['y']  #  value of list does not change
۴ change (list)
```

```
۱ def func (s,i) :
۲     s = s + ' x '
۳     i = i + 1
۴ strval = ' hello '
۵ intval = 5
۶ change ( strval, intval )
۷ # value of strval and inval does not change
```

روش‌های ارسال پارامتر

- در زبان‌هایی که نوع آرگومان‌ها در برابر نوع پارامترها بررسی نمی‌شود، ممکن است خطاهایی در برنامه نویسی رخ دهد که برای پیدا کردن مشکل باشد.
- به عنوان مثال فرض کنید بررسی نوع [1] در پارامترهای توابع وجود نداشته باشد و یک آرگومان float به یک تابع که ورودی double می‌گیرد ارسال شود. اگر ارسال با مقدار باشد، مشکلی رخ نمی‌دهد اما اگر ارسال با ارجاع باشد، تابع در یک متغیر ۴ بایتی ۸ بایت داده ذخیره می‌کند. در خلال اجرای برنامه هیچ خطایی مشاهده نخواهد شد، اما نتیجه برنامه ممکن است با نتیجه دلخواه نابرابر باشد و در چنین مواردی پیدا کردن خطا بسیار مشکل خواهد بود.

[1] type checking

زیر برنامه به عنوان پارامتر

- در برخی مواقع نیاز است یک زیر برنامه به عنوان آرگومان به یک زیر برنامه دیگر ارسال شود.
- به عنوان مثال تابعی را در نظر بگیرید که یک تابع به عنوان ورودی دریافت می‌کند و انتگرال آن را در یک بازهٔ معین محاسبه می‌کند. اگر نتوانیم تابع به عنوان پارامتر ارسال کنیم برای هر تابع ریاضی باید یک تابع انتگرال‌گیر نیز پیاده سازی کنیم، که باعث افزایش هزینه زمانی برای نوشتن برنامه و همچنین افزایش حافظه مورد نیاز برای برنامه می‌شود.
- در زبان سی و سی++، اشاره‌گر به توابع [1] می‌توانند به عنوان آرگومان به توابع ارسال شوند. از آنجایی که در تعریف اشاره‌گر به تابع نوع‌های ورودی و خروجی تابع یا به عبارت دیگر امضای تابع مشخص می‌شود، بنابراین بررسی نوع در زمان کامپایل می‌تواند صورت بگیرد.
- برای مثال `float (*pfun)(int , double)` یک اشاره‌گر در ورودی یک تابع باشد، هر تابعی با این امضا می‌تواند به عنوان آرگومان یک تابع مد نظر ارسال شود.

[1] function pointer

زیر برنامه به عنوان پارامتر

- در زبان پایتون توابع می‌توانند به عنوان آرگومان به توابع دیگر ارسال شوند. به جای تابع در آرگومان می‌توان همچنین از یک عبارت لامبدا استفاده کرد. برای مثال دیدیم که توابع نگاشت و فیلتر و کاهش نیاز به دریافت تابع به عنوان ورودی دارند.

زیر برنامه به عنوان پارامتر

- مقدار بازگشتی توسط یک تابع در زبان سی می‌تواند هر نوعی به غیر از آرایه و تابع باشد. اما در زبان پایتون هر نوعی می‌تواند توسط یک تابع بازگردانده شود. یک تابع می‌تواند تابع نیز بازگرداند.
- در بیشتر زبان‌ها یک تابع می‌تواند تنها یک مقدار بازگرداند. با استفاده از نوع چندتایی در زبان پایتون می‌توان از یک مقدار (به عنوان مقادیر یک چندتایی) بازگرداند.

زیر برنامه به عنوان پارامتر

- یک تابع در برخی زبان‌ها مانند سی++ می‌تواند سربارگذاری [1] شود بدین معنا که می‌توان تعدادی تابع هم‌نام تعریف کرد که پارامترهایی از نوع‌های متفاوت دریافت کنند. همچنین در زبان سی++ می‌توان عملگرها را سربارگذاری کرد، بدین معنی که یک عملگر با توجه به عملوندهای آن عملیاتی متفاوت انجام دهد.
- برای مثال در سی++ می‌توانیم عملگر + را برای اشیای یک کلاس به صورت زیر تعریف کنیم :

```
Complex operator +(Complex c1, Complex c2){
    return Complex ( c1.r + c2.r, c1.i + c2.i) ;
}
```

[1] overload

زیر برنامه‌های عمومی

- یکی از معیارهای یک زبان برای ارزیابی، قابلیت آن برنامه برای نوشتن کد به صورت مختصر و بهینه است. در یک زبان هر چقدر عملیات بیشتری در حجم کمتر بتوان نوشت، راندمان برنامه نویسی در آن افزایش می‌یابد.
- برای مثال اگر در یک زبان قابلیتی وجود داشته باشد که عملیات یکسان بر روی نوع‌های متفاوت بتوانند تنها توسط یک تابع پشتیبانی شوند، راندمان برنامه نویسی افزایش می‌یابد.
- در برنامه نویسی شیءگرا خواهیم دید که انواع متفاوت که همگی از یک خانواده هستند می‌توانند توسط قابلیت چند ریختی [1] به یک تابع ارسال شوند.

[1] polymorphism

زیر برنامه‌های عمومی

- قابلیت دیگری که توسط برخی زبان‌ها پشتیبانی می‌شود، دریافت ورودی با نوع پارامتری است، بدین معنا که یک زیر برنامه علاوه بر دریافت ورودی‌ها به عنوان پارامتر، نوع ورودی‌ها را نیز به عنوان پارامتر دریافت می‌کند.
- در زبان سی++ این قابلیت توسط قالب‌ها [1] پشتیبانی می‌شود.
- در برخی زبان‌ها مانند پایتون که نوع متغیرها به طور صریح توصیف نمی‌شود، توابع همگی یک نوع عمومی دریافت می‌کنند.

[1] templates

زیر برنامه‌های عمومی

- برای مثال برای مقایسه دو مقدار از یک نوع عمومی در زبان سی++ می‌توانیم تابعی به صورت زیر بنویسیم :

```
۱ template  < class Type>
۲ Type max (Type first , Type second) {
۳       return first > second ? first : second ;
۴ }
```

- چنانچه می‌خواستیم این تابع را بدون برنامه نویسی عمومی تعریف کنیم، باید به ازای هر نوع یک تابع جدید تعریف می‌کردیم. به علاوه ممکن بود پس از تعریف این تابع در یک کتابخانه محاسباتی، یک برنامه نویس به مقایسه نوع‌هایی نیاز داشت که در زمان توسعه کتابخانه موجود نبودند. با استفاده از برنامه نویسی عمومی استفاده کننده کتابخانه محاسباتی نوع‌های جدید را می‌تواند به عنوان پارامتر به توابع کتابخانه ارسال کند، اما بدون برنامه نویسی عمومی لازم بود استفاده کننده کتابخانه از توسعه دهندهٔ کتابخانه بخواهد توابع جدید با نوع‌های جدید را به کتابخانهٔ محاسباتی اضافه کند.

پیوند زیر برنامه

- به عملیات فراخوانی و بازگرداندن مقدار یک زیر برنامه و اتصال آن به جریان برنامه اصلی پیوند زیر برنامه [1] گفته می‌شود.

- عملیاتی که در پیوند زیر برنامه، برای انتقال متغیرها انجام می‌شود، باید توصیف شده و توسط طراح زبان پیاده سازی شوند. برای مثال متغیرهای محلی یک زیر برنامه بر روی پشته تخصیص داده می‌شوند و وضعیت اجرای [2] برنامه قبل از فراخوانی برنامه ذخیره می‌شود تا پس از فراخوانی ادامه پیدا کند. وضعیت اجرای برنامه شامل مقادیر رجیسترها و بیت‌های حالت پردازنده می‌شود. همچنین باید کنترل اجرا به زیر برنامه داده شود و اطمینان حاصل شود که پس از اجرای زیر برنامه کنترل به دستور بعد از فراخوانی بازمی‌گردد. اگر امکان پیاده سازی زیر برنامه‌های تودرتو وجود داشته باشد، باید اطمینان حاصل شود که زیر برنامه‌ها به متغیرهای غیر محلی به درستی دسترسی دارند. در بازگرداندن مقدار ، مقدار خروجی تابع و همچنین متغیرهایی که با ارجاع یا حالت خروجی ارسال شده‌اند باید در دسترس برنامه فراخوانی کننده قرار بگیرند.

---

[1] subprogram linkage

[2] execution status

پیوند زیر برنامه

- نحوهٔ پیوند یک زیر برنامهٔ ساده را بررسی می‌کنیم. در این زیر برنامهٔ ساده زیر برنامه تودرتو نمی‌تواند وجود داشته باشد و همچنین همهٔ متغیرها ایستا هستند. نسخه‌های اولیهٔ زبان فورترن به این شکل بودند.

- در زمان فراخوانی اعمال زیر باید انجام شوند :
  ۱. وضعیت اجرای برنامه جاری ذخیره شود.
  ۲. آرگومان‌ها به پارامترها ارسال شوند.
  ۳. آدرس بازگشت به زیر برنامهٔ فراخوانی شونده ارسال شود.
  ۴. کنترل اجرا به زیر برنامهٔ فراخوانی شونده ارسال شود.

- در زمان خاتمه و بازگشت زیر برنامه عملیات زیر باید انجام شوند :
  ۱. اگر پارامتری با حالت خروجی (برای مثال ارسال با ارجاع) وجود دارد، مقدار آن‌ها باید در دسترس برنامه فراخوانی کننده قرار بگیرد.
  ۲. اگر زیر برنامه دارای خروجی است، خروجی آن باید در دسترس برنامه فراخوانی کننده قرار بگیرد.
  ۳. وضعیت اجرای فراخوانی کننده به حالت قبل از فراخوانی باز می‌گردد.
  ۴. کنترل اجرا به فراخوانی کننده باز می‌گردد.

پیوند زیر برنامه

- بنابراین در اجرای یک زیر برنامه، اطلاعات در مورد وضعیت فراخوانی کننده، پارامترها، آدرس بازگشت و مقادیر بازگشتی از زیر برنامه باید ذخیره شوند. این اطلاعات به همراه متغیرهای محلی و کد زیر برنامه مجموعه‌ای از اطلاعات مورد نیاز یک زیر برنامه را تشکیل می‌دهند.
- یک زیر برنامه از دو بخش تشکیل شده است : کد زیر برنامه و متغیرهای محلی و داده‌های مورد نیاز.
- داده‌های یک زیر برنامه به همراه متغیرهای محلی رکورد فعالسازی [1] یک زیر برنامه گفته می‌شوند زیرا داده‌هایی که توصیف شدند در زمان فعالسازی زیر برنامه مورد نیازند. یک نمونه از رکورد فعالسازی [2] در واقع یک مثال یا یک نمونه از مقادیر از رکورد فعالسازی است.
- از آنجایی که در یک زیر برنامهٔ ساده همه مقادیر و داده‌ها ثابت هستند به طور ایستا می‌توانند تخصیص داده شوند. توجه کنید که امکان فراخوانی بازگشتی با استفاده از این روش وجود ندارد.

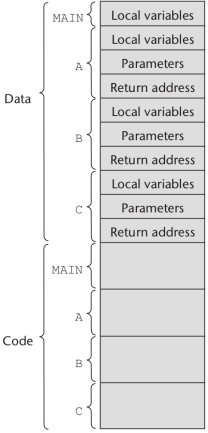
---

[1] activation record

[2] activation record instance

پیوند زیر برنامه

- شکل زیر رکورد فعالسازی برای سه زیر برنامه A و B و C را نشان می‌دهد.



پیوند زیر برنامه

- برنامه اجرایی شکل قبل توسط پیوند دهنده یا لینکر [1] ایجاد می‌شود.
- وقتی لینکر اجرا می‌شود، ابتدا همهٔ فایل‌های کامپایل شدهٔ زیر برنامه‌ها در حافظه قرار می‌گیرند. سپس برای هر فراخوانی زیر برنامه، آدرس آن در حافظه باید در مکان فراخوانی قرار بگیرد. همچنین به ازای هر فراخوانی زیر برنامه در درون یک زیر برنامه آدرس مکان ورود به زیر برنامه‌ها توسط لینکر در برنامه اجرایی قرار می‌گیرد.

[1] linker

پیوند زیر برنامهٔ حاوی متغیر پشته

- حال پیوند زیر برنامه را در زبانی بررسی می‌کنیم که متغیرهای محلی به صورت پویا بر روی پشته قرار می‌گیرند. یکی از مهم‌ترین مزیت‌های حافظه پویا بر روی پشته این است که توسط آن می‌توان توابع بازگشتی را پیاده سازی کرد.
- کامپایلر باید کدی تولید کند که تخصیص و آزاد سازی متغیرهای محلی را انجام دهد.
- همچنین در این حالت ممکن است تابعی به صورت بازگشتی فراخوانی شود و بنابراین بیش از یک نمونه از یک زیر برنامه در هر مرحله می‌تواند فعال باشد.
- در زبان‌هایی که دارای متغیرهای پویای بر روی پشته هستند نمونه رکورد فعالسازی به صورت پویا ساخته می‌شود.

پیوند زیر برنامهٔ حاوی متغیر پشته

- یک رکورد فعالسازی شامل آدرس بازگشت، لینک پویا [1] ، پارامترها و متغیرهای محلی زیر برنامه می‌شود.
- آدرس بازگشت اشاره‌گری است که به دستور بعد از فراخوانی زیر برنامه اشاره می‌کند.
- لینک پویا اشاره‌گری است که به نمونه رکورد فعالسازی برنامه یا زیر برنامه فراخوانی کننده اشاره می‌کند. در زبان‌هایی که حوزهٔ تعریف پویا دارند، از لینک پویا برای دسترسی به متغیرهای محلی توابع فراخوانی کننده یک تابع استفاده می‌شود. در زبان‌هایی که حوزهٔ تعریف ایستا دارند از این لینک برای دنبال کردن خطاها و گزارش آن توسط کامپایلر به برنامه نویس استفاده می‌شود.
- همچنین مقدار آرگومان‌ها در زمان فراخوانی در رکورد فعالسازی فراخوانی کننده قرار دارند که مقادیر آنها باید در مقدار پارامترهای تابع فراخوانی شوند کپی شود.

[1] dynamic link

پیوند زیر برنامهٔ حاوی متغیر پشته

- تابع زیر را در نظر بگیرید.

```
۱ void sub (float total, int part) {
۲      int list [5];
۳      float sum;
۴ }
```

پیوند زیر برنامهٔ حاوی متغیر پشته

- رکورد فعالسازی این تابع در زیر نشان داده شده است.

Local	`sum`
Local	`list [4]`
Local	`list [3]`
Local	`list [2]`
Local	`list [1]`
Local	`list [0]`
Parameter	`part`
Parameter	`total`
Dynamic link	
Return address	

پیوند زیر برنامهٔ حاوی متغیر پشته

- از آنجایی که آخرین تابع فراخوانی شونده اولین تابعی است که باید اجرا شود، طبیعی است که ساختار حافظه پشته باشد. به این پشته، پشته زمان اجرا [1] گفته می‌شود. هر زیر برنامه، رکورد فعالسازی خود را بر روی پشته می‌سازد و هر فراخوانی زیر برنامه رکورد مربوط به خود را دارد.
- اشاره‌گر محیطی [2] به رکورد فعالسازی آخرین زیر برنامه اشاره می‌کند. در هر بار فراخوانی یک زیر برنامه، زیر برنامه فراخوانی شده آدرس رکورد فعال سازی فراخوانی کننده را در لینک پویای خود نگهداری می‌کند و آدرس اشاره‌گر محیطی را به آدرس رکورد فعالسازی خود تغییر می‌دهد.

[1] run-time stack
[2] environment pointer (EP)

پیوند زیر برنامهٔ حاوی متغیر پشته

- بنابراین عملیات زیر از سوی فراخوانی کننده باید انجام شوند.
  ۱. یک نمونه رکورد فعالسازی ساخته می‌شود.
  ۲. وضعیت اجرای برنامه فعلی ذخیره می‌شود.
  ۳. آرگومان‌ها به پارامترها ارسال می‌شوند.
  ۴. آدرس بازگشت به فراخوانی شونده ارسال می‌شود.
  ۵. کنترل اجرای برنامه به فراخوانی شونده داده می‌شود.

پیوند زیر برنامهٔ حاوی متغیر پشته

- زیر برنامه فراخوانی شونده در شروع عملیات زیر را انجام می‌دهد.
  ۱. آدرس اشاره‌گر محیطی را در لینک پویا ذخیره می کند و آدرس رکورد فعالسازی خود را به اشاره‌گر محیطی می‌دهد.
  ۲. فضا برای متغیرهای محلی خود بر روی پشته تخصیص می‌دهد.
- زیر برنامه فراخوانی شونده در پایان کار عملیات زیر را انجام می‌دهد.
  ۱. پارامترها با حالت خروجی را در آرگومان‌ها کپی می‌کند.
  ۲. اگر زیر برنامه مقدار خروجی باز می‌گرداند، مقادیر را به تابع فراخوانی کننده بازمی‌گرداند.
  ۳. مقدار اشاره‌گر محیطی را برابر با مقدار لینک پویا قرار می‌دهد.
  ۴. وضعیت اجرای فراخوانی کننده را بازمی‌گرداند.
  ۵. کنترل اجرا را به فراخوانی کننده بازمی‌گرداند.

پیوند زیر برنامهٔ حاوی متغیر پشته

- وقتی اجرای یک زیر برنامه به پایان می‌رسد رکورد فعالسازی آن تخریب می‌شود بنابراین متغیرهای محلی آن دیگر در دسترس نیستند.

پیوند زیر برنامهٔ حاوی متغیر پشته

- حال برنامه زیر را در نظر بگیرید.

```
۱  void fun1(float r) {
۲      int s, t;
۳      ... // Point  1
۴      fun2(s);
۵      ...
۶  }
۷  void fun2(int x) {
۸      int y;
۹      ... // Point 2
۱۰     fun3(y);
۱۱     ...
۱۲ }
۱۳ void fun3(int q) {
۱۴     ... // Point 3
۱۵ }
```

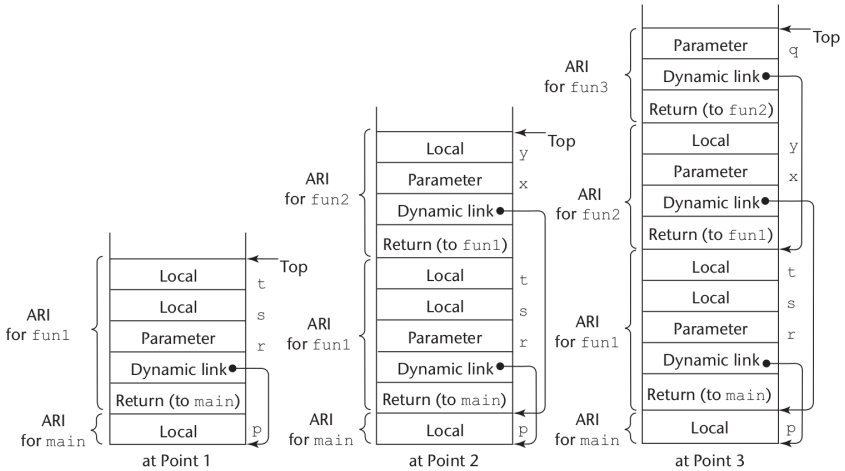
پیوند زیر برنامهٔ حاوی متغیر پشته

- تابع main تابع fun1 را فراخوانی می‌کند. سپس fun1 ، تابع fun2 و fun2 ، تابع fun3 را فراخوانی می‌کند.

```
۱۶ void main() {
۱۷     float p;
۱۸     ...
۱۹     fun1(p);
۲۰     ...
۲۱ }
```

پیوند زیر برنامهٔ حاوی متغیر پشته

- محتوای پشته برای این برنامه در زیر نشان داده شده است.



ARI = activation record instance

پیوند زیر برنامهٔ حاوی متغیر پشته

- مجموعه لینک‌های پویا، زنجیرهٔ پویا [1] یا زنجیرهٔ فراخوانی [2] نیز نامیده می‌شود. زنجیرهٔ فراخوانی تاریخچه فراخوانی توابع را نمایش می‌دهد.
- هر کدام از متغیرهای محلی در پشته یک آفست محلی [3] دارند که مکان آنها نسبت به شروع رکورد فعالسازی را مشخص می‌کند.

[1] dynamic chain
[2] call chain
[3] local offset

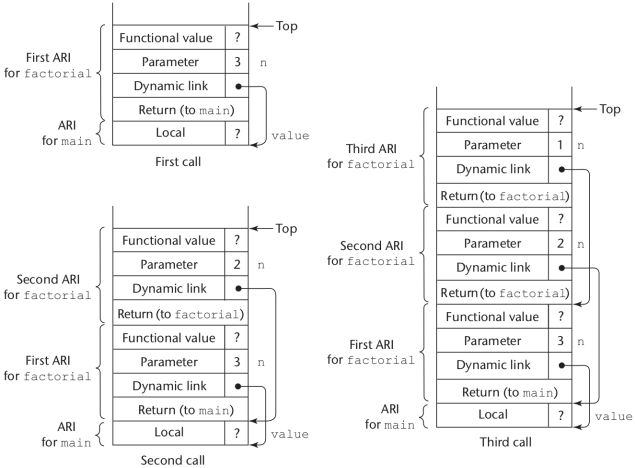
پیوند زیر برنامهٔ حاوی متغیر پشته

- حال برنامه زیر را در نظر بگیرید که در آن فراخوانی بازگشتی وجود دارد.

```
۱  int factorial(int n) {
۲      // Point 1
۳      if (n <= 1)
۴          return 1;
۵      else
۶          return (n * factorial(n - 1));
۷      // Point 2
۸  }
۹  void main() {
۱۰     int value;
۱۱     value = factorial(3);
۱۲     // Point 3
۱۳ }
```

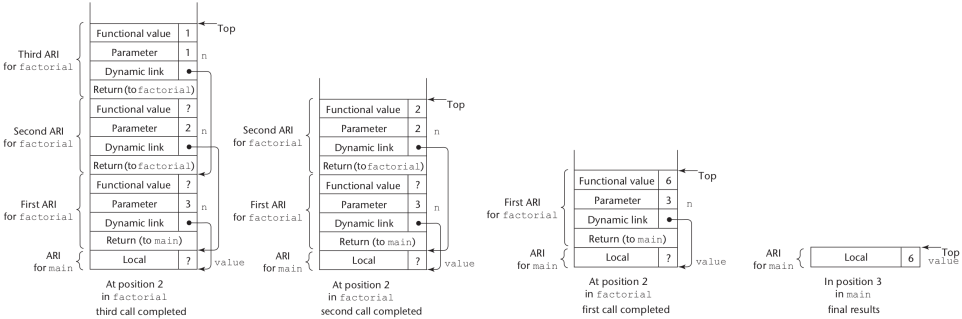
پیوند زیر برنامهٔ حاوی متغیر پشته

- محتوای پشته این برنامه تا رسیدن به نقطه اول در شکل زیر نشان داده شده است.



پیوند زیر برنامهٔ حاوی متغیر پشته

- محتوای پشته برای این برنامه تا رسیدن به نقطه دوم در شکل زیر نشان داده شده است.



زیر برنامه‌های تودرتو

- برخی از زبان‌ها مانند پایتون و روبی اجازه می‌دهند زیر برنامه‌های تودرتو ساخته شوند.
- در پیاده سازی این زبان‌ها علاوه بر لینک پویا که برای دنبال کردن زیر برنامه‌ها به کار می‌رود، یک مقدار دیگر به نام لینک ایستا [1] نیز استفاده می‌شود که برای دنبال کردن دسترسی متغیرها در توابع تودرتو به کار می‌رود. به لینک ایستا، اشاره‌گر حوزه تعریف ایستا [2] نیز گفته می‌شود که در واقع به رکورد فعال سازی زیر برنامه پدر اشاره می‌کند.
- همچنین زنجیرهٔ ایستا [3] به دنبالهٔ لینک‌های ایستا که به یکدیگر متصل شده‌اند گفته می‌شود.
- برای پیدا کردن مقادیر متغیرهای غیر محلی، باید زنجیرهٔ ایستا را دنبال کرد و در زیر برنامهٔ پدر یا اجداد مقدار متغیر غیر محلی را جستجو کرد.

---

[1] static link
[2] static scope pointer
[3] static chain

زیر برنامه‌های تودرتو

- برنامه زیر را در نظر بگیرید.

```
۱  function main(){
۲      var x;
۳      function bigsub() {
۴          var a, b, c;
۵          function sub1 {
۶              var a, d;
۷              ...
۸              a = b + c; // Point 1
۹              ...
۱۰         } // end of sub1
```

زیر برنامه‌های تودرتو

```
۱۱        function sub2(x) {
۱۲            var b, e;
۱۳            function sub3() {
۱۴                var c, e;
۱۵                ...
۱۶                sub1();
۱۷                ...
۱۸                e = b + a; // Point 2
۱۹            } // end of sub3
۲۰            ...
۲۱            sub3();
۲۲            ...
۲۳            a = d + e; // Point 3
۲۴        } // end of sub2
```

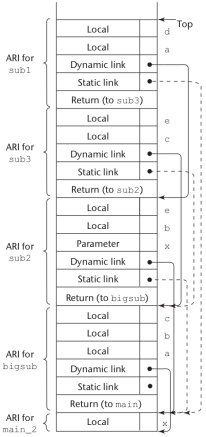
زیر برنامه‌های تودرتو

```
        ...
        sub2(7);
        ...
    } // end of bigsub
    ...
    bigsub();
    ...
} // end of main
```

- در این برنامه تابع main تابع bigsub را فراخوانی می‌کند، سپس bigsub تابع sub2 و sub2 تابع sub3 و sub3 تابع sub1 را فراخوانی می‌کند.
- از طرفی sub1 و sub2 در تابع bigsub تعریف شده‌اند و sub3 در تابع sub2 .

زیر برنامه‌های تودرتو

- پشته فراخوانی برای این برنامه در شکل زیر نشان داده شده است.



بلوک‌ها

- در برخی از زبان‌ها مانند سی می‌توان بلوک‌هایی [1] برای حوزهٔ تعریف متغیرها تعریف کرد.
- برای مثال با بازکردن یک آکولاد بلوک جدیدی ساخته می‌شود که متغیرهای آن بلوک فقط درون بلوک تعریف شده‌اند و خارج از آن قابل دسترسی نیستند.
- بلوک‌ها را نیز می‌توان توسط زنجیره‌های ایستا پیاده سازی کرد. هر بلوک می‌تواند به عنوان یک زیر برنامه بدون پارامتر در نظر گرفته شود.

[1] blocks