بازگشتی

تابع بازگشتی تابعی است که در بدنه اش دستوری دارد که خودش را فراخوانی می کند. توابع بازگشتی برای نگهداری حالت قبلی خود از پشته مکرر استفاده می کنند .

Call stack
عملکردهای call stack
ساختار call stack
بازگشتی
بازگشتی در مقایسه با غیر بازگشتی
سرریزی پشته
نمونه هائی از توابع بازگشتی

تابع یا زیربرنامه بخش جداگانه ای از کد برنامه است که می تواند توسط یک نام صدا زده شود. به هر زیربرنامه ممکن است پارامترهایی ارسال شود و هر تابع می تواند یک مقدار را برگرداند. وقتی تابعی فراخوانی می شود مقدار پارامترهای تابع در جائی باید ذخیره شود تا تابع بتواند به آنها دسترسی پیدا کند .

Call stack

اکثر کامپایلرها برای فراخوانی و برگشت از زیربرنامه call stack را پیاده سازی می کنند. Call stack یا run-time stack یک پشته است که اطلاعاتی درباره زیربرنامه فعال یک برنامه را نگهداری می کند. زیربرنامه فعال زیربرنامه ای است که فراخوانی شده است اما هنوز اجرایش تمام نشده است .

وقتی زیربرنامه ای فراخوانی می شود، قبل از اینکه کنترل اجرای برنامه به آدرس زیربرنامه پرش کند آدرس دستورالعمل بعدی (دستورالعملی که درحافظه بعد از دستور فراخوانی قرار دارد) درجائی باید ذخیره شود که هنگام برگشت از زیربرنامه از آن استفاده می شود. این آدرس را آدرس برگشتی (return addresses) می نامند .

معماری که بر اساس پشته است آدرس برگشتی را به عنوان نقطه برگشت در پشته اضافه می شود. هر بار که زیربرنامه ای فراخوانی می شود آدرس برگشتی در پشته push می شود. هنگام برگشت از زیربرنامه آدرس برگشتی از پشته pop شده و کنترل برنامه به آن آدرس پرش می کند و اجرای برنامه از بعد از دستور فراخوانی ادامه پیدا می کند .

به دلیل استفاده از پشته یک زیربرنامه می تواند خودش یا زیربرنامه های دیگر را صدا بزند.

در زبان های سطح بالا call stack معمولا از برنامه نویس مخفی است .درمقابل در زبان اسمبلی نیاز است خود برنامه نویس با پشته درگیر شود .

عملکردهای call stack

هدف اصلی یک call stack نگهداشتن آدرس برگشتی هر زیربرنامه فعال است .اما بسته به زبان، سیستم عامل و محیط سخت افزاری ممکن است عملکردهای اضافی دیگری هم داشته باشد نظیر :

ذخیره آدرس های برگشتی
برای هر برنامه یک پشته درنظرگرفته می شود. وقتی زیربرنامه ای در برنامه فراخوانی می شود آدرس دستورالعمل بعد از عبارت فراخوانی (آدرس برگشتی) در پشته قرار می گیرد. زیربرنامه می تواند به صورت بازگشتی باشد هر بار که زیربرنامه خودش را صدا می زند آدرس برگشتی در پشته ذخیره می شود .
ذخیره متغیرهای محلی
متغیرهائی که درون زیربرنامه تعریف می شوند متغیرهای محلی نامیده می شوند. متغیرهای محلی تنها درون زیربرنامه فعل شناخته شده هستند و بعد از اتمام زیربرنامه مقدار آنها در حافظه باقی نمی ماند. اغلب مناسب است که فضائی در پشته به آنها

اختصاص داده شود که سریع تر از تخصیص فضای آزاد heap به آنها است. هر زیربرنامه فعال فضای جداگانه خودش را در پشته برای داده های محلی دارد .

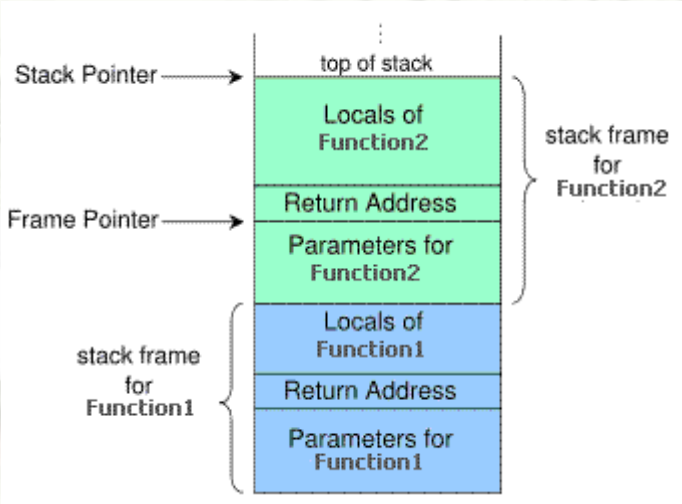
ارسال پارامتر

مقادیر پارامترهای موردنیاز زیربرنامه ها هنگام فراخوانی به آنها داده می شوند. معمولا فضای از call stack برای ذخیره مقدار این پارامترها اختصاص داده می شود. هر فراخوانی به زیربرنامه مقادیر مختلفی از پارامترها را خواهد داشت و فضای جداگانه ای در پشته به آنها داده می شود.

ساختار call stack

یک call stack از stack frame ها یا activation record ها تشکیل شده است. فریم پشته اطلاعات زیربرنامه را نگه می دارد. هر فریم پشته مربوط به یک فراخوانی زیربرنامه ای است که هنوز تمام نشده است .

مثال. فرض کنید تابع function2 اکنون در حال اجرا است و توسط function1 فراخوانی شده است. وضعیت پشته می تواند به شکل زیر باشد .



فریمی که در بالای پشته است مربوط به زیربرنامه ای است که اکنون در حل اجرا است. هر فریم ممکن است دربرگیرنده متغیرهای محلی، آدرس برگشتی و مقدار پارامترهای زیربرنامه باشد. فریم های پشته هم اندازه نبوده و زیربرنامه های مختلف فریم های متفاوتی دارند .

پشته توسط ثبات stack pointer دسترسی می شود که بالای پشته را مشخص می کند .

بازگشتی

بازگشتی اجازه بیان راه حل یک مسئله را به طور مختصر و مفید می دهد .مسئله ای که به صورت بازگشتی حل می شود باید بتواند به مسائل کوچک تر تقسیم بشود و حل مسائل کوچک به همان روش مسئله بزرگ قابل انجام باشد .مسئله کوچک تر به مسئله کوچک تری شکسته می شود تا سرانجام به کوچک ترین اندازه مسئله برسد که base case نامیده می شود که می تواند بدون استفاده از بازگشتی حل شود .

یک مثال متعارف الگوریتم بازشگتی ضابطه تابع فاکتوریل f(n) است :

$$f(n) = \begin{cases} 1 & \text{if } n = 0 \\ n \times f(n-1) & \text{if } n > 0 \end{cases}$$

مقدار تابع برای f(3) به صورت زیر محاسبه می شود:

f(3) = 3 * f(3 − 1)
= 3 * f(2)
= 3 * 2 * f(2 − 1)
= 3 * 2 * f(1)
= 3 * 2 * 1 * f(1 − 1)
= 3 * 2 * 1 * f(0)
= 3 * 2 * 1 * 1
= 6

یک الگوریتم بازگشتی توسط تابع بازگشتی پیاده سازی می شود. تابع بازگشتی (recursive function) تابعی است که خودش را فراخوانی می کند .

تابع بازگشتی مشابه توابع دیگر کار می کند. برای هر فراخوانی بازگشتی یک فریم جدید از تابع در پشته ایجاد می شود .

مثل(C) . تابع بازگشتی محاسبه فاکتوریل یک عدد صحیح.

```
int Factorial (int x)
{
  if (x<= 1)
    return 1;
  else
    return x * Factorial (x-1);
}
```

همان تابع به صورت غیر بازگشتی به صورت زیر نوشته می شود. توجه کنید که به د و متغیر موقت نیاز دارد .

```
int Factorial (int x)
{
   int i, temp;
   for ( i=1; i<=x; i++)
      temp*=i;
   return temp;
}
```

مثل (C). تابع دیگر که زیبائی بازگشتی را بیشتر نشان می دهد الگوریتم اقلیدسی برای محاسبه بزرگترین مقسوم علیه مشترک (greatest common divisor)دو عدد صحیح است. الگوریتم بازگشتی آن در زبان C به صورت زیر است :

```
int GCD (int x, int y)
{
   if (y == 0)
      return x;
   else
      return GCD (y, x % y);
}
```

الگوریتم غیر بازگشتی آن به صورت زیر است. مشاهده می شود که الگوریتم غیر بازگشتی احتیاج به یک متغیر موقت دارد و حتی با دانستن الگوریتم اقلیدسی درک فرآیند تابع مشکل است .

```
int GCD(int x, int y)
{
while (y != 0) {
  int r = x % y;
  x = y;
  y = r;
  }
  return x;
}
```

هر تابع بازگشتی می تواند توسط یک پشته به تابع غیر بازگشتی تبدیل شود .علت اینکه توابع بازگشتی ممکن به سختی ردیابی شوند احتمالا نداشتن دانش کافی درباره نحوه کار پشته است .

بازگشتی در مقایسه با غیر بازگشتی

برای اینکه بازگشتی موفق باشد مسئله نیاز است یک زیرساختار بازگشتی داشته باشد. راه حل بعضی از مسائل به طور ذاتی بازگشتی است چون احتیاج به نگداری حالت قبلی دارند. الگوریتم پیمایش درخت(tree traversal)، تابع اکرمن (Ackermann) و الگوریتم های تقسیم و غلبه مانند مرتب سازی سریع (Quicksort) همگی به صورت بازگشتی هستند. همه این الگوریتم ها می توانند به صورت غیربازگشتی با کمک پشته هم پیاده شوند اما نیاز به پشته مزیت راه حل غیر بازگشتی را از بین می برد .

تابع غیر بازگشتی احتمالا در عمل کمی سریعتر از نسخه بازگشتی آن اجرا می شود چون تابع غیر بازگشتی سربار فراخوانی تابع (function-call)را به اندازه تابع بازگشتی ندارد و این سربار در بعضی زبان ها نسبتا بالا است .

یک دلیل دیگر برای ترجیح غیربازگشتی به بازگشتی این است که فضای پشته قابل دسترس کمتر از فضای قابل دسترس در حافظه آزاد heap است. و الگوریتم های بازگشتی تمایل به فضای پشته بیشتری نسبت به غیر بازگشتی دارند .

سرریزی پشته

وقتی اشاره گر پشته به انتهای پشته می رسد پشته سرریز می شود. دلیل معمول سرریزی پشته فراخوانی مکرر یا تعداد زیاد متغیرهای محلی توابع بازگشتی است. اگر یک تابع بی نهایت بار خودش را صدا بزند در هر فراخوانی یک فریم پشته اضافه می شود و در یک نقطه پشته دیگر جا ندارد و سرریز می شود و خطای stack overflow رخ می دهد .

نمونه هائی از توابع بازگشتی

مثل .(C)تابع بازگشتي ضرب.

```
int Mul (int a , int b)
{
  if (b == 1)
    return a;
  else
    return a+Mul (a,b-1);
}
```

مثل .(Pascal)تابع بازگشتی فیبوناچی.

```
Function Fibonacci ( n: Integer ):Integer;
Begin
  If (n=1) or (n=2) Then Fib:=1
  Else Fibonacci:= Fibonacci (n-2)+ Fibonacci (n-1);
End;
```

مثل .(Pascal)تابع بازگشتي اکرمن (Ackermann's function) تابعی است که مقدار آن به سرعت رشد می کند .

```
Function Ackerman ( a,b: Integer ):Integer;
Begin
  If (a<0) and (b<0) Then Ackerman:=0
  Else If a=0 Then Ackerman:=b+1
  Else If b=0 Then Ackerman:= Ackerman (b-1,1)
  Else Ackerman:= Ackerman (a-1,Ack(a,b-1));
End;
```