بنام کسی که آفرید از عدم به انسان عطا کرد فکر و قلم

موضوع مقاله : Recursive

تاریخ: ۱۳۸۹/۰۸/۱۵

ارائه دهنده: زهرا هادی

نام استاد : مهندس محمد سلیمی

WWW.SALIMITEACH.COM

توابع بازگشتی:

امکان فراخوانی یک تابع در تابع دیگر همواره وجود دارد.خاصیت مهم اینجاست که در زبان C# میتوان درون یک تابع، خود تابع را بنا با لیست آرگومانهای جدید فراخوانی کرد. از خاصیت بالا برای انجام کارهایی که به صورت تکرار محاسبات مراحل قبلی صورت میگیرند، میتوان استفاده نمود.

در توابع بازگشتی، توجه به دو نقطه بسیار مهم است:

۱) نقطه بازگشت = نقطه بازگشت جایی است که باید خود تابع با آرگومان لیست جدید فراخوانی شود. یعنی به نقطه ی بست نرسیده ایم.

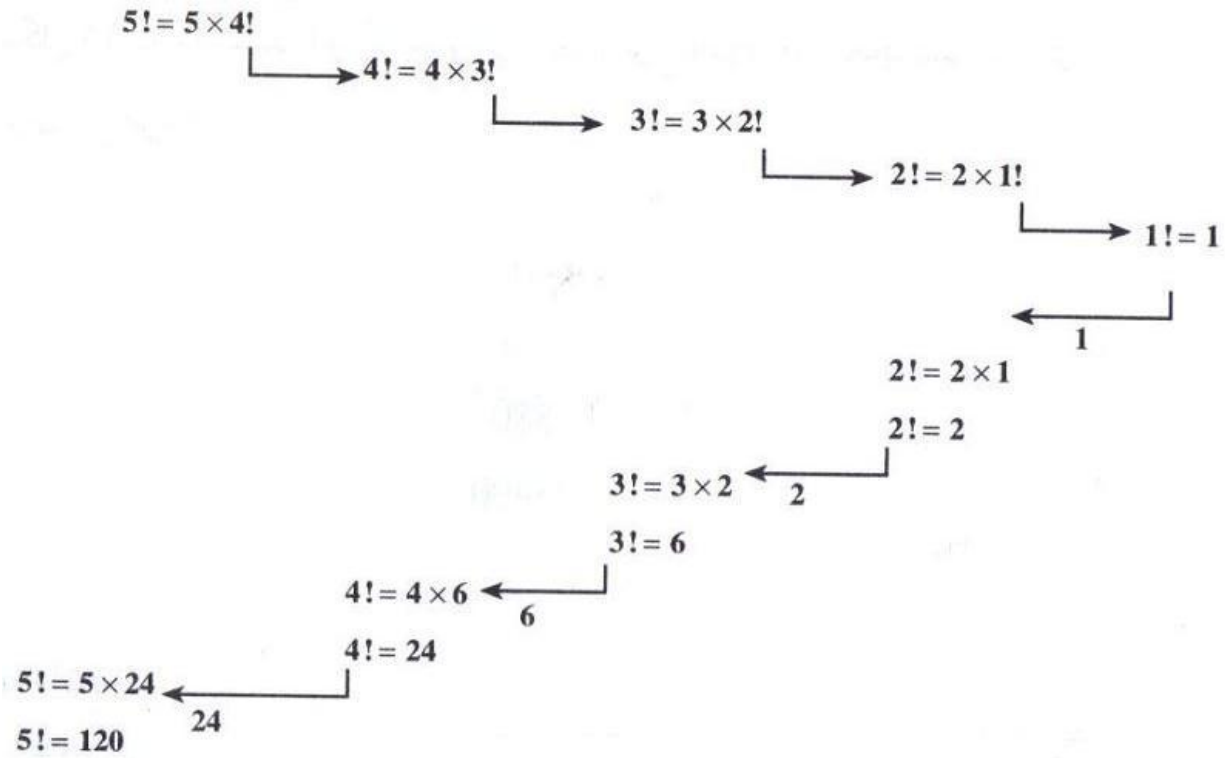
۲) نقطه بن بست = نقطه بن بست جایی است که در آنجا باید روند فراخوانی را که انجام داده ایم، بصورت بازگشت به عقب انجام دهیم تا جواب مطلوب حاصل شود.

مثال: تابع بدست آوردن فاکتوریل عدد از نوع بازگشتی:

محاسبه فاکتوریل عدد

Factorial

```
//recursive function calculates n!
static int FactorialRecursive(int n)
{
    if (n <= ۱) return ۱;
    return n * FactorialRecursive(n - ۱);
}
```



کد برنامه قبلی بدون تابع بازگشتی

```
//iterative function calculates n!
static int FactorialIterative(int n)
{
    int sum = ١;
    if (n <= ١) return sum;
    while (n > ١)
    {
        sum *= n;
        n--;
    }
    return sum;
}
```

۶!=۷۲۰

۷!=۵۰۴۰

۸!=۴۰۳۲۰

۹!=۳۶۲۸۸۰

۱۰!=۳۶۲۸۸۰۰

نقاط جالب

۱. سعی نکنید که در موقعیت های حساس از recursive استفاده کنید.
۲. راه حل elegant (یک برنامه با کمترین مقدار حافظه اصلي طراحي يک برنامه کارا که با کم کردن تعداد دستورالعملهاي بکار برده شده براي انجام کارهاي گوناگون از حداقل ممکن حافظه اصلي استفاده کند،ظريف ،زيبا،با سليقه ،پربراز،برازنده) همیشه بهترین راه حل نیست.
۳. اگر نیاز داری به استفاده از recursive حتما برنامه رو به‌نحوی طراحی کنی که برای هر چه نزدیک تر شدن به dynamic programming

با چند مثال بحث را پی می گیریم

- سری فیبوناچی
- تعیین بزرگ ترین مقسوم علیه مشترک بین دو عدد
- برج هانوی

سری فیبوناچی

Fibonacci

```
//--------------- iterative version ------------------
--
static int FibonacciIterative(int n)
{
    if (n == ۰) return ۰;
    if (n == ۱) return ۱;

    int prevPrev = ۰;
    int prev = ۱;
    int result = ۰;
```

```
    for (int i = ٢; i <= n; i++)
    {
        result = prev + prevPrev;
        prevPrev = prev;
        prev = result;
    }
    return result;
}

//--------------- naive recursive version -------------
--------
static int FibonacciRecursive(int n)
{
    if (n == ٠) return ٠;
    if (n == ١) return ١;

    return FibonacciRecursive(n - ١) +
FibonacciRecursive(n - ٢);
}

//--------------- optimized recursive version ---------
------------
static Dictionary<int> resultHistory = new
Dictionary<int>();

static int FibonacciRecursiveOpt(int n)
{
    if (n == ٠) return ٠;
    if (n == ١) return ١;
    if (resultHistory.ContainsKey(n))
        return resultHistory[n];

    int result = FibonacciRecursiveOpt(n - ١) +
FibonacciRecursiveOpt(n - ٢);
    resultHistory[n] = result;
```

```
    return result;
}
```

تعیین بزرگ ترین مقسوم علیه مشترک بین دو عدد

ابتدا محاسبه ب م م با روش نردبانی را یادآوری می کنیم

خارج قسمت تقسیم عدد اول (مثلاa) بر عدد دوم (مثلاb) را بالای عدد دوم و باقیمانده را جلوی عدد دوم در جدول می نویسیم

Step۱ :ابتدا چک می کنیم که عدد دوم صفر نشده باشد

Step۲ : aرا می گذاریم عدد دوم b را می گذاریم باقیمانده

Step۳: a را بر b تقسیم می کنیم اگر باقیمانده صفر شد a بزرگ ترین مقسوم علیه مشترک بین دو عدد است

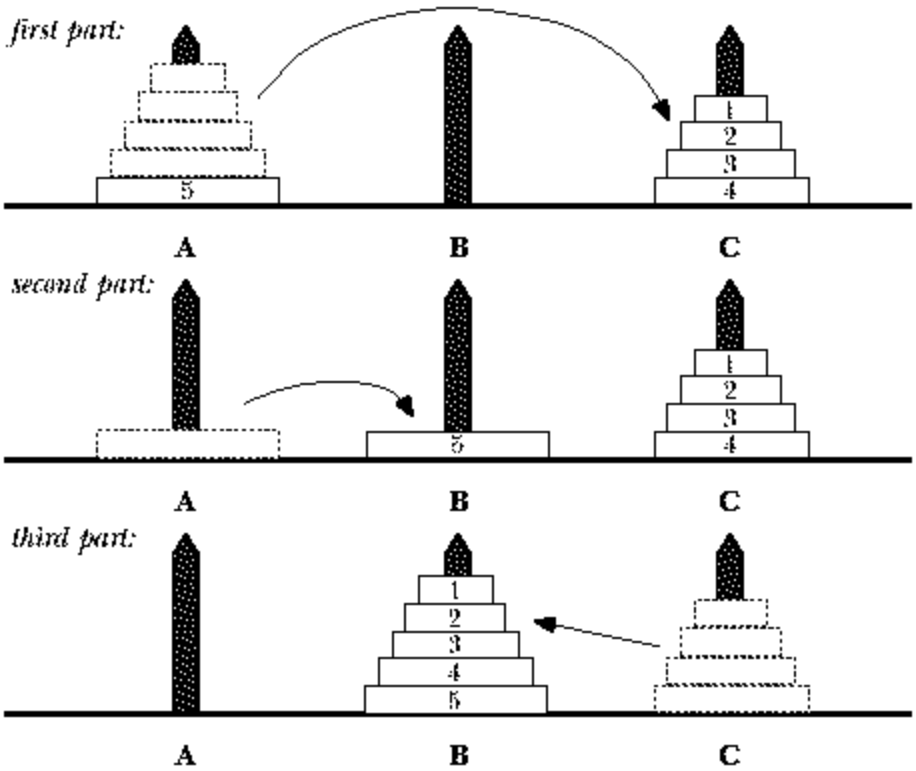
```
static  int gcd(int a, int b)

        {
            int item;
            if (b == ۰)
            {
                item = a;
            }
            else
            {
                item = gcd(b, a % b);
            }
            return item;
        }
```

مسئله برجهای هانوی (Towers of Hanoi)

هر برنامه نویسی باید به نحوی با مسائل کلاسیک دست وپنجه نرم کرده باشد . یکی از معروفترین مسائل کلاسیک ، مسئله برجهای هانوی می باشد. طبق افسانه ای در معبدی در شرق دور، کاهنان معبد ی تعدادی دیسک را از یک ستون به ستون دیگر جا به جا م ی کردند . ستون اول در ابتدا دارای ۶۴ دیسک با اندازه های مختلف می باشد، که بزرگترین دیسک در پایین ستون و کوچکترین دیسک در بالای ستون قرار دارد. کاهنان باید همه دیسکها را از یک ستون به ستون دوم منتقل می کردند. با این شرط که در هر بار جا به جایی تنها یک دیسک منتقل شود و نیز دیسک بزرگتری روی دیسک کوچکتر قرار نگیرد . ضمناً ستون سوم ی به عنوان ستون کمکی در اختیار آنها می باشد. گویند هنگامی که کاهنان معبد همه ۶۴ دیسک را با روش گفته شده از ستون اول به ستون دوم منتقل کنند جهان به پایان می رسد.

برای راحتی کار کاهنان و برای اینکه دچار اشتباه و دوباره کاری در انتقال نشوند می خواهیم برنامه ای بنویسیم که ترتیب انتقال دیسکها را چاپ کند.

برای نوشتن این برنامه ، مسئله را باید با دید بازگشتی نگاه کنیم . انتقال **n** دیسک را به شیوه زیر انجام می دهیم :



۱- ابتدا **n-۱** دیسک را از ستون اول به ستون دوم به کمک ستون سوم منتقل کن.

۲- دیسک آخر (بزرگترین دیسک) را از ستون اول به ستون سوم منتقل کن.

۳- **n-۱** دیسک قرار گرفته در ستون دوم را به کمک ستون اول به ستون سوم منتقل کن.

مراحل انجام کار هنگام انتقال آخرین دیسک یعنی وقتی که **n=۱** می باشد، یعنی حالت پایه به اتمام می رسد. در حالت **n=۱** یک دیسک بدون کمک ستون کمکی به ستون دیگر منتقل می شود.تابع بازگشتی مورد استفاده برای حل مسئله برجهای هانوی را با چهار آرگومان می نویسیم.

۱- تعداد دیسکها ۲- ستون مبدأ ۳- ستون کمکی ۴- ستون مقصد

```
#include
int hanoi(int, char, char, char);
int main( )
    { int disks;
        cout<<"Moving disks form tower A to C."<
        cout<<"How many disks do you want to move?";
        cin>>disks;
        cout<<hanoi(disks,'A','B','C')<
        return ۰;
    }
int hanoi(int n, char first, char help, char second)
    {
        if (n == ۱) {
            cout << "Disk " << n << " from tower " <<
first
            << " to tower " << second << endl;
                    }
        else {
            hanoi(n-۱, first, second, help);
            cout << "Disk " << n << " from tower " <<
first
            << " to tower " << second << endl;
            hanoi(n-۱, help, first, second);
            }
        return ۰;
    }
```