الخوارزميات العودية

المحاضرة الخامسة
م. لمى السبع

الخوارزميات العودية

- نقول عن برنامج جزئي (تابع) أنه عودي، عندما يستدعي نفسه. وبالتالي يتم تنفيذ هذا التابع بشكل متكرر.
- من المفيد في بعض المسائل استخدام التوابع العودية عوضاً عن الحلقات التكرارية مثلاً مسائل الذكاء الصنعي.
- كل تابع عودي بحاجة لشرط توقف لإنهاء تكرار الاستدعاء.

فائدة الخوارزميات العودية

- تستخدم في المسائل المعقدة حيث تقوم بتقسيم المسألة الكبيرة لمسائل جزئية أصغر حتى تصل لحالة بدائية.
- ثم تعود خطوة بخطوة تعوض القيم الناتجة إلى أن تصل للحالة للمسألة الأصلية . فنكون حصلنا على الناتج النهائي.

❖ **ملاحظة:**

- يجب أن يكون عمق العودية (أي عدد تكرار الاستدعاءات العودية لنفس التابع) منتهيا وصغير.

سلبيات الخوارزميات العودية

١. الاستدعاءات العودية تسبب ضياع في الوقت وتستهلك ذاكرة إضافية (لأن كل استدعاء يعتبر تابع جديد يتم حجز متحولاته المحلية ووسطائه من جديد.)

٢. بعض اللغات البرمجة لا تقبل تعريف التوابع العودية مثال : لغة الآلة ، Fortran، Cobol

أنواع الخوارزميات العودية

التوابع ذات العودية المباشرة	التوابع ذات العودية غير المباشرة
هي توابع تحوي استدعاءً صريحاً لنفسها . مثال :	هي توابع تحوي استدعاءً لتوابع أخرى و التي بدورها تستدعي التابع الأب . مثال : إذا كان لدينا التابع

int A(void) {
.....
A( ); // direct recursive call
.....
}
	void B(void) {
	.....
	A( );
	.....
	}
	
	int A(void) {
	.....
	B( ) ; // indirect recursive call
	.....
	}

تابع حساب العاملي لعدد

من المعروف أن قانون العاملي يُحسب كالتالي : $n! = n \cdot (n-1) \cdot (n-2) \cdot \ldots\ldots \cdot 2 \cdot 1$

و لكن يمكن كتابته بشكل آخر : $n! = n \cdot (n-1)!$

$$(n-1) \cdot (n-2)!$$

$$(n-2) \cdot (n-3)!$$

$$1 \cdot (0!)$$

إذاً تتكرر عملية حساب العاملي لعدد ما و حيث أن هذا العدد بدأ من القيمة n و يتناقص في كل مرة حتى أن يصل إلى الصفر , علماً أن الصفر أصغر قيمة مقبولة لحساب العاملي لها . لذلك يمكن هنا تطبيق مفهوم العودية لأننا نكرر عمليات حساب العاملي لأعداد متناقصة حتى أن نصل إلى الحالة البدائية .

لنكتب تابع حساب العملي لعدد بطرقتين (تكرارية و عودية) :

تابع حساب العاملي لعدد

حساب العاملي لعدد تكرارياً	حساب العاملي لعدد عودياً

`long fact(int n)`
`{  long f=1;`
`  if ((n==1)||(n==0)) return 1;`
`  else`
`  { for (int i=1;i<=n;i++)`
`  f*=i;`
`  return f; }`
`}`
	`long fact(int n)`
	`{  if ((n==1)||(n==0))`
	`  return 1;`
	`  else`
	`  return n*fact(n-1);`
	`}`

حساب قيمة متتالية فيبوناتشي عند عدد ما

إن الشكل العام لمتتالية فيبوناتشي هو :

$$F_n = F_{n-1} + F_{n-2} \quad ; \; n \geq 2 \; (F_0 = 0 \; , \; F_1 = 1)$$

و بالتالي لدينا مثلاً :

F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	.....
0	1	1	2	3	5	8	13	.....

إذاً لدينا حالتين بدائيتين (0 و 1) بينما من أجل أي عدد آخر لا يمكن حساب القيمة إلا بالاعتماد على القيم المسبقة , هذا يعني أنه لدينا تكرار حساب تابع فيبوناتشي و لكن في كل مرة لأعداد أصغر حتى أن نصل إلى أبسط قيم ممكنة (أي 0 و 1) .

و يكتب التابع العودي كما يلي :

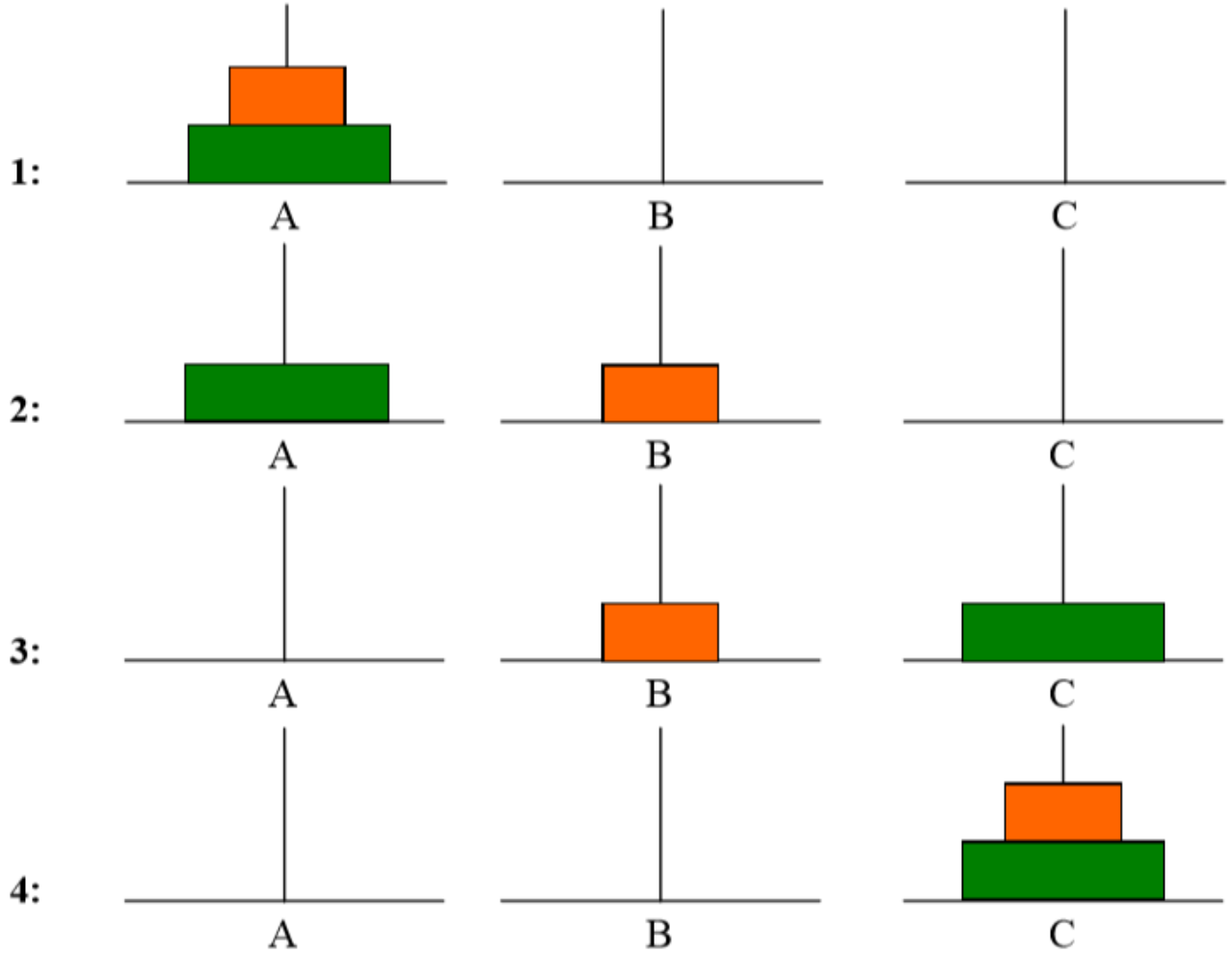
```
int fib(int n)
{ if (n==0) return 0;
  else
  if (n==1) return 1;
  else      return fib(n-1)+ fib(n-2);
}
```

مسألة أبراج هانوي

- أبراج هانوي هي أحجيةٌ وضعها العالم لوكاس عام ١٨٨٣ وتنصُّ على ما يلي:
- لدينا عدد n من الأقراص، متوضعّة على عمودٍ بترتيبٍ منتَظَمٍ بناءً على أحجامِها، بحيثُ يتوضّع القرصُ لأكبر في الأسفل والقرصُ الأصغر في الأعلى، نريد نقلَ هذه الأقراص من العمود الموجودة فيه إلى عمودٍ آخرَ باستخدامِ عمودٍ ثالثٍ للمساعدَة، لكنْ ضمنَ شروطٍ محددةٍ:
- نقلِ قرصٍ واحدٍ فقط في النقلةِ الواحدة
- عدمِ وجوبِ توضُّع أحدِ الأقراص فوق قرصٍ أصغرَ منه، أي يجبُ المحافظةُ على الترتيب حتى عند الحركات الوسيطة للنقل (باستخدام العمودِ الثالث).

مثال توضيحي

فمثلاً من أجل $n = 2$ لدينا :



الكود البرمجي

```
0 references
static void Main(string[] args)
{

        Console.WriteLine( "Enter n= ");
    int n = int.Parse(Console.ReadLine());
    hanoi(n, 'A', 'B', 'C');
}
3 references
static void  hanoi(int n, char A, char B, char C)
    {
        if (n == 1)
        Console.WriteLine( A + " --> " + C );
      else {
        hanoi(n - 1, A, C, B);
        Console.WriteLine(A + " --> " + C);
        hanoi(n - 1, B, A, C);
    }
    }
```

الخرج

حيث يعطي تنفيذ البرنامج السابق النتائج التالية (من أجل قرص أو قرصين أو ثلاثة أقراص) :

n = 1	n = 2	n = 3
A → C	A → B A → C B → C	A → C A → B C → B A → C B → A B → C A → C