# **Bài 6. Hàm (tiếp)**

**Mục tiêu:**

1. *Luyện tập việc dùng mảng trong hàm*
2. *Luyện tập đệ quy đơn giản.*

***Giới hạn:*** *không dùng các thư viện stl (chẳng hạn vector, algorithm)*

**Yêu cầu nộp bài (một trong hai lựa chọn)***:*

1. *Toàn bộ phần A. code A1, A5-8, kết quả các bài A2-6 ghi vào file readme.txt.*
2. *Ít nhất 03 bài phần C.*

*Bài này đánh giá theo cố gắng làm việc. Bạn cần làm đủ. Chương trình của bạn có thể chưa chạy đúng, thậm chí chưa chạy, nhưng vẫn có thể được tính là có làm và đạt.   
Bạn có thể làm vượt yêu cầu để bù điểm cho các bài tập trước. Viết vào* ***readme.txt*** *trong thư mục* ***BT06*** *nếu có nhu cầu này (chú ý đặt tên file chính xác).*

***Lưu ý: chép bài hoặc cho chép bài sẽ dẫn đến trượt môn học!***

## Thực hành

1. ***Mảng là tham số của hàm.*** Hãy viết một chương trình thử nghiệm để xem mảng được truyền vào hàm theo cơ chế gì.  
   Gợi ý:viết một hàm f nhận tham số là một mảng. Từ bên trong hàm, in ra địa chỉ của biến mảng và địa chỉ phần tử đầu tiên trong mảng. Viết hàm main với một mảng A được khởi tạo sẵn. Tại hàm main, in ra địa chỉ của biến mảng và địa chỉ phần tử đầu tiên trong mảng A. Gọi hàm f với tham số là mảng A.   
   Nhận xét, so sánh kết quả in ra từ main và từ f.
2. ***Lỗi đệ quy thiếu base case***. Hãy chạy thử hàm sau, bạn thấy hiện tượng gì? Giải thích lí do.

double H(int N) {   
 return H(N-1) + 1.0/N;   
}

1. ***Lỗi công thức đệ quy***. Hãy chạy thử hàm sau, bạn thấy hiện tượng gì? Giải thích lí do.

double H(int N) {

if (N == 1) return 1.0;  
 return H(N) + 1.0/N;   
}

1. ***Đệ quy quá sâu***. Hãy chạy thử hàm sau với N = 5000, bạn thấy hiện tượng gì? Nếu chưa thấy gì thì tăng giá trị của N cho đến khi thấy chương trình trục trặc. Giải thích lí do.

double H(int N) {

if (N == 1) return 0.0;  
 return H(N-1) + 1.0/N;   
}

1. ***Đệ quy tính toán quá nhiều***. Cho cài đặt sau của hàm tính dãy fibonacii.Hãy thử gọi với n đủ lớn để thấy chương trình chạy lâu. Cài lại bằng vòng lặp để so sánh thời gian chạy.

long F(int n) {   
 if (n == 0) return 0;   
 if (n == 1) return 1;   
 return F(n-1) + F(n-2);   
}

1. ***Dùng mảng địa phương trong hàm đệ quy***. Thử nghiệm một hàm đệ quy trong đó khai báo một mảng kich thước lớn. Hãy thử xem hàm đó đệ quy sâu bao nhiêu tầng thì bị lỗi tràn bộ nhớ.
2. ***Duyệt hoán vị.*** Cài chương trình theo thuật toán đã hướng dẫn tại bài giảng. Input: một xâu kí tự không chứa kí tự trắng. Output: tất cả các hoán vị của xâu kí tự, mỗi hoán vị nằm trên một dòng.
3. ***Duyệt tổ hợp.*** Cài chương trình theo thuật toán đã hướng dẫn tại bài giảng. Input: số nguyên dương N. Output: liệt kê tất cả các tập con của tập [1..N], mỗi tập con nằm trên 1 dòng với dạng tương tự [2,3]

## Câu hỏi

1. Nếu bỏ qua các vấn đề về bộ nhớ và thời gian chạy, liệu có tình huống nào mà vòng lặp là lựa chọn duy nhất để giải quyết mà không thể dùng đệ quy?  
   Không. Tất cả các giải pháp dùng vòng lặp đều có thể được cài bằng đệ quy, mặc dù dùng đệ quy có thể tốn quá nhiều bộ nhớ.
2. Có tình huống nào mà đệ quy là lựa chọn duy nhất (không thể dùng vòng lặp để cài đặt thuật toán)?  
   Không, cách trình biên dịch xử lý các hàm đệ quy chính là một cách dùng vòng lặp để thay thế tất cả các kiểu đệ quy. Các bạn sẽ học về phương pháp này ở môn Cấu trúc dữ liệu và giải thuật.
3. Nên dùng chiến lược nào hơn, đệ quy hay vòng lặp?

Tùy tình huống, hãy chọn cách nào đơn giản, dễ hiểu, và hiệu quả.

1. Cần đề phòng những gì khi viết hàm đệ quy?
2. Không chạy chương trình, hãy tính xem output của lời gọi hàm ex233(6) là gì.

void ex233(int n) {  
 if (n <= 0) return;  
 cout << n << " ";  
 ex233(n-2);  
 ex233(n-3);  
 cout << n << " ";  
}

1. Hàm đệ quy sau có vấn đề gì?

long factorial(int n) {

long temp = factorial(n-1) \* n;  
 return (n <= 1) ? 0 : temp;

}

1. Cho hàm đệ quy sau:

int mystery(int a, int b) {  
 if (b == 0) return 0;  
 if (b % 2 == 0) return mystery(a+a, b/2);  
 return mystery(a+a, b/2) + a;  
}

Lời gọi hàm mystery(2,25) và mystery(3,11) cho kết quả gì? Thực ra mystery(a,b) tính cái gì với a,b>0?  
Câu hỏi tương tự, nhưng sửa code của mystery thay + bằng \* và thay return 0 bằng return 1.

## Bài tập

Tất cả các bài trong phần này đều cần dùng hàm một cách thích hợp, không hàm nào dài quá 15 dòng.

1. ***Sắp hậu.*** Trên một bàn cờ vua kích thước NxN, hãy tìm tất cả các cách xếp N con hậu sao cho không có hai con hậu nào đe dọa nhau (cùng hàng/cột/đường chéo).   
   Input: N   
   Output: in tất cả các cách xếp ra màn hình. Mỗi cách ở dạng bảng vuông ‘\*’ và ‘.’.  
   Gợi ý: Mô hình hóa một cách sắp xếp hậu bằng một mảng một chiều, trong đó phần tử thứ i có giá trị bằng tọa độ cột của con hậu tại hàng i.  
   Công thức đệ quy: Hồi quy bài toán xếp k hậu vào các hàng từ 1 đến k về bài toán xếp k-1 hậu vào các hàng từ 1 đến k-1. Cụ thể, để duyệt các cách xếp hậu vào các hàng từ 1 đến k, ta xét từng cách xếp con hậu tại hàng k, với mỗi cách, gọi đệ quy bài toán xếp hậu vào các hàng từ 1 đến k-1.
2. ***Hoán vị độ dài k***. Nhập 2 số nguyên N và K, in ra tất cả các từ độ dài K gồm các chữ cái khác nhau trong phạm vi N chữ cái tiếng Anh bắt đầu từ ‘a’.
3. ***Tổ hợp độ dài k***. Nhập 2 số nguyên N và K, in ra tất cả các tập hợp gồm K phần tử gồm các chữ cái khác nhau trong phạm vi N chữ cái tiếng Anh bắt đầu từ ‘a’.
4. ***Ma trận Hadamard.*** Dùng chiến lược đệ quy, hãy viết chương trình Hadamard nhận input là một số nguyên n, tạo output là một ma trận kích thước N\*N với N = 2^n, với pattern dưới đây (n = 1, 2, 3, và 4). Ma trận Hadamard kích thước 2N\*2N có đặc điểm sau: ba góc phần tư là ba ma trận Hadamard kích thước N\*N, góc phần tư phải dưới là đảo lại của ma trận Hadamard kích thước N\*N. Dùng các kí tự ‘.’ và ‘o’ để in bảng ra màn hình.   
   *Bạn tự giới hạn kích thước N tuỳ theo cách bạn vẽ ma trận ra màn hình, sao cho một hàng không vượt quá chiều ngang màn hình (vì nếu vượt quá và bị xuống dòng thì trông rất xấu).*

| 2-by-2 Hadamard plot | 4-by-4 Hadamard plot | 8-by-8 Hadamard plot | 16-by-16 Hadamard plot |
| --- | --- | --- | --- |

1. ***Partition***. Viết chương trình nhập 1 số nguyên dương và in ra tất cả các dãy số nguyên dương có tổng bằng số đó. Ví dụ:  
   Input:   
   4  
   Output:   
   4  
   3 1  
   2 2  
   2 1 1  
   1 1 1 1
2. ***Hoán vị Johnson-Trotter***. Viết chương trình nhập 1 số nguyên dương n và liệt kê tất cả các hoán vị của các số từ 1 đến n theo trình tự Johnson-Trotter: hai hoán vị liền nhau chỉ khác nhau bởi một lần đổi chỗ. Ví dụ với input n = 3, output gồm các hoán vị theo trình tự như sau:  
   123  
   132  
   312  
   321  
   231  
   213

Xem thuật toán tại <https://en.wikipedia.org/wiki/Steinhaus%E2%80%93Johnson%E2%80%93Trotter_algorithm>