

KỸ THUẬT LẬP TRÌNH

Phạm Thế Bảo

<http://www.math.hcmuns.edu.vn/~ptbao/KTLT/>
ptbao@hcmuns.edu.vn

Trường Đại học Khoa học Tự nhiên Tp.HCM

NỘI DUNG

1. Thiết kế chương trình
2. Mảng một chiều, hai chiều và nhiều chiều
3. Con trỏ
4. Chuỗi ký tự
5. Cấu trúc
6. Tập tin
7. Đệ quy
8. Một số thuật toán thông dụng

Cách tính điểm

- Điểm thực hành 50% tổng điểm
- Điểm lý thuyết 50% tổng điểm
- Điểm cộng thêm 10 – 20% tổng điểm

Tài liệu tham khảo

1. *Quách Tuấn Ngọc*, Ngôn Ngữ Lập Trình C. Nhà Xuất Bản Giáo Dục, 1998.
2. *Mark Allen Weiss*, Efficient C programming.. Prentice Hall, 1998.
3. *Yale N. Patt, Sanjay J. Patel*, Introduction to Computing System, from Bits and Gates to C and Beyond.. McGrawHill, 1999.
4. *Trần Đan Thư*, Giáo trình lập trình C (Tập 1 & 2). NXB ĐH QG TP HCM –2003.
5. *Nguyễn Thanh Thủy*, Nhập môn lập trình ngôn ngữ C. NXB KH-KT – 2005.
6. *Phạm Văn Át* - Kỹ thuật lập trình C cơ sở và nâng cao. NXB KH-KT- 2006.



Ebook

1. **Beginning C From Novice to Professional.**
2. **C Primer Plus 5th Edition.**
3. **Mastering Algorithms with C.**
4. **Practical C Programming.**
5. **Expert C Programming - Deep C Secrets.**
6. **The Complete Reference.**
7. **C Reference Card (ANSI).**

Website

1. <http://www.congdongviet.com>
2. <http://www.cprogramming.com/>
3. <http://www.programmingtutorials.com/c.aspx>
4. <http://www.codeguru.com/>
5. <http://www.thecodeproject.com/>
6. <http://c4swimmers.net>
7. <http://www.vocw.edu.vn/>
8. <http://www.google.com.vn/>

CÁC PHƯƠNG PHÁP GIẢI QUYẾT BÀI TOÁN TRÊN MÁY TÍNH

Phạm Thế Bảo

Trường Đại học Khoa học Tự nhiên Tp.HCM

Phân loại

1. Phương pháp trực tiếp
2. Phương pháp gián tiếp hoặc tìm kiếm lời giải

Phương pháp trực tiếp

- Xác định trực tiếp được lời giải qua một thủ tục tính toán (công thức, hệ thức, định luật, ...) hoặc qua các bước căn bản để có được lời giải.
- Việc giải quyết vấn đề trên máy tính chỉ là thao tác lập trình hay là sự chuyển đổi lời giải từ ngôn ngữ tự nhiên sang ngôn ngữ máy tính → kỹ thuật lập trình trên máy tính.
- Có ba loại cơ bản:
 - Loại thứ nhất, dùng để biểu diễn cho các bài toán đã có lời giải chính xác bằng một công thức toán học nào đó.
$$1+2+\dots+n = \frac{n(n+1)}{2}$$

ví dụ: tính tổng n số nguyên dương.
 - Loại thứ hai, biểu diễn cho các bài toán có công thức giải gần đúng (công thức tính sin, cos, giải phương trình siêu việt, ...).
ví dụ: giải phương trình bậc 2
 - Loại cuối cùng, biểu diễn các lời giải không tường minh bằng kỹ thuật đệ quy.

Chuyển đổi dữ liệu bài toán thành dữ liệu chương trình

Nguyên lý 1: Dữ liệu của bài toán sẽ được biểu diễn lại dưới dạng các biến của chương trình thông qua các quy tắc xác định của ngôn ngữ lập trình cụ thể

1. Biến - phương tiện biểu diễn dữ liệu của chương trình
2. Thay đổi giá trị của biến - lệnh gán
3. Kiểu dữ liệu
4. Hằng số
5. Cấu trúc một chương trình

Chuyển đổi quá trình tính toán của bài toán thành các cấu trúc của chương trình

- Nguyên lý 2 (Định lý Bohn-Jacopini): Mọi quá trình tính toán đều có thể mô tả và thực hiện dựa trên ba cấu trúc cơ bản: tuần tự, rẽ nhánh và lặp.
 1. Cấu trúc tuần tự
 2. Cấu trúc rẽ nhánh
 1. Rẽ nhánh có điều kiện: if (condition)
 - rẽ nhánh đơn: if ()
 - rẽ nhánh đôi: if () ... else ...
 2. Rẽ nhiều nhánh: case
 3. Rẽ nhánh không có điều kiện: LABEL và GOTO
 3. Cấu trúc lặp:
 1. Lặp xác định
 2. Lặp không xác định

Phân chia bài toán ban đầu thành những bài toán nhỏ hơn

- Nguyên lý 3: Mọi bài toán lớn đều có thể giải quyết bằng cách phân chia thành những bài toán nhỏ hơn
 1. Thủ tục và hàm - phương pháp phân chia chương trình thành những chương trình con.
 2. Biến cục bộ và biến toàn cục
 3. Tham số - dữ liệu đầu vào/đầu ra của hàm

Biểu diễn tính toán không tường minh bằng đệ quy

- Nguyên lý 4: quá trình đệ quy trong máy tính không đơn giản như các biểu thức quy nạp trong toán học
- Sẽ trình bày sau này.

Phương pháp gián tiếp

- Được sử dụng khi chưa tìm ra lời giải chính xác của vấn đề.
- Đây là cách tiếp cận chủ yếu của loài người từ xưa đến nay.
- Lời giải trực tiếp bao giờ cũng tốt hơn, nhưng không phải lúc nào cũng có

Phân loại phương pháp gián tiếp

1. Phương pháp thử - sai
 1. Thử - sai hệ thống
 2. Thử - sai phân lớp
 3. Thử - sai ngẫu nhiên
2. Phương pháp Heuristic
3. Phương pháp trí tuệ nhân tạo

Phương pháp thử - sai

Thomas Edison – phát biểu cách tìm một cây kim trong một đồng rom: *“trong khi chưa nghĩ ra được một cách thật hay thì cứ việc rút từng cọng rom cho đến khi rút được cây kim”*.


Phương pháp này dựa trên 3 nguyên lý:

1. Nguyên lý vét cạn (duyệt toàn bộ): liệt kê tất cả các trường hợp xảy ra và xem xét chúng.

Ví dụ: liệt kê tất cả số nguyên tố từ m đến n.

2. Nguyên lý ngẫu nhiên: dựa trên việc thử một số khả năng được chọn một cách ngẫu nhiên trong tập khả năng (thường rất lớn, nếu áp dụng nguyên lý toàn bộ sẽ tốn nhiều thời gian). Khả năng tìm lời giải đúng (hoặc gần đúng) sẽ phụ thuộc vào chiến lược chọn ngẫu nhiên và một số điều kiện cụ thể.

Ví dụ: kiểm tra chất lượng trong quá trình sản xuất của một đoàn kiểm tra. Một lô hàng có 1000 thùng, chọn ngẫu nhiên 10 thùng, mỗi thùng có 24 sản phẩm, chọn ngẫu nhiên 5 sản phẩm, ...



Nguyên lý được phát triển thành phương pháp Monté-Carlos. Ngày càng nguyên lý ngẫu nhiên càng phát triển mạnh mẽ, trong số đó có một phương pháp nổi bật là phương pháp Genetic.

3. Nguyên lý mê cung: nguyên lý này được áp dụng khi chúng ta không biết chính xác "hình dạng" của lời giải, mà phải xây dựng lời giải dần qua từng bước, giống như tìm đường ra khỏi mê cung.

Thử sai - hệ thống

1. Nguyên lý vét cạn toàn bộ: muốn tìm cây kim trong đồng rơm, hãy lần lượt rút từng cọng rơm đến khi rút được cây kim.

Thuật giải: gọi D là không gian bài toán (tập tất cả khả năng xảy ra), $D = \{(x_1, x_2, \dots, x_n) / x_i \in D_i \text{ với } D_i \text{ là tập hữu hạn có } m_i \text{ phần tử}\}$.

gọi $f: D \rightarrow \{\text{true}, \text{false}\}$ là quy tắc xác định lời giải.

Ví dụ: một đàn gà và một bầy chó có tổng cộng N chân, đàn gà đông hơn bầy chó M con. Hỏi có bao nhiêu gà và chó?

- 
2. Nguyên lý mắt lưới: lưới bắt cá chỉ bắt được những con cá có kích thước lớn hơn kích thước mắt lưới.

Ví dụ:

Tìm nghiệm phương trình trong một đoạn

Khử nhiễu trong ảnh

3. Nguyên lý mê cung: Muốn thoát khỏi mê cung thì phải biết quay lui và biết đánh dấu những nơi đã đi qua.

Ví dụ:

Tìm đường đi ngắn nhất

Thử - sai phân lớp

1. Nguyên lý chung về giảm độ phức tạp của thử - sai: thu hẹp tập trường hợp trước và trong khi duyệt, đồng thời đơn giản hóa tối đa điều kiện chấp nhận một trường hợp.
2. Quy tắc:
 1. đơn giản điều kiện: tránh tính lại trong vòng lặp và thừa kế kết quả tính toán của bước trước: tổ hợp chỉnh hợp, heap sort,
 2. Kỹ thuật cầm canh: mã đi tuần,
 - số âm đầu tiên trong mảng: điều kiện `while(x[i]>0&& i<=n) do` cách khác `a[n+1]=-1; while(x[i]>0) do`
3. Nguyên lý thu gọn không gian tìm kiếm: loại bỏ những trường hợp hoặc nhóm trường hợp chắc chắn không dẫn đến lời giải.

- Quy tắc rút gọn:

1. Dựa trên đánh giá toàn cục: tìm điều kiện để rút gọn tập khả năng đề cử trong một bước xây dựng một thành phần.

Ví dụ: tìm tổ hợp chập n của k .

2. Dựa trên đánh giá cục bộ: xây dựng phép kiểm tra đơn giản để nhanh chóng loại bỏ được các khả năng cho thành phần $x[i]$ mà không phải xây dựng toàn bộ $n-i$ thành phần còn lại của lời giải.

Ví dụ: cho sáu số tự nhiên $A = \{1, 7, 2, 9, 3, 5\}$. Tìm dãy con của A sao cho tổng các phần tử trong dãy con bằng 8.

4. Nguyên lý đánh giá nhánh cận: nhánh có chứa quả phải nặng hơn trọng lượng của quả.

Ví dụ: bài toán người du lịch.

5. Quay lui không dùng đệ quy

6. Phương pháp sinh lời giải

Phương pháp Heuristic

- Trong nhiều bài toán dùng phương pháp thử - sai sẽ dẫn đến số lượng thử quá lớn không chấp nhận được.
- Heuristic chính là ước lượng về khả năng dẫn đến lời giải của một trạng thái: phương pháp vét cạn nhưng có thêm tri thức đi kèm, tối ưu cục bộ, nguyên lý hướng đích, nguyên lý sắp thứ tự,
- ví dụ:
Một em bé bị lạc đường về nhà, em nhớ nhà mình cao nhất trong khu vực, em sẽ tìm đến tòa nhà cao nhất trong vùng em thấy, rồi lại tiếp tục, ...
Giải phương trình bậc 2, đoán nghiệm theo Vi-ét

Tìm kiếm theo chiều sâu và chiều rộng

- Là thử - sai theo nguyên lý mê cung hay chính là thử - sai kết hợp lẫn ngược.
- Ngược với tìm kiếm theo chiều sâu, tìm kiếm theo chiều rộng mang hình ảnh của vết dầu loang.

—————> Giải thuật A*

Phương pháp trí tuệ nhân tạo

- "Dạy" máy tính để có "trí thông minh" như con người bắt chước khả năng "suy luận" của con người.

ví dụ: bài toán đong nước, có 3 bình A, B, và C có dung tích 5, 8, và 13 lít. Làm sao đong được 11 lít nước trong bình C? Bình C ban đầu đầy nước.



Một số phương pháp chuyển giao tri thức

1. Biểu diễn tri thức
2. Hệ chuyên gia
3. Máy học



Mảng - Array

Phạm Thế Bảo

Trường Đại học Khoa học Tự nhiên Tp.HCM

Mảng – Array

- Một số tính chất
- Khai báo mảng trong C
- Truy xuất các thành phần
- Truyền tham số kiểu mảng cho hàm
- Một số thao tác cơ sở
- Mảng nhiều chiều

Mảng – Một số tính chất

- Mảng là một kiểu dữ liệu có cấu trúc do người lập trình định nghĩa
- Dùng biểu diễn các đối tượng dữ liệu ở dạng một dãy các thành phần có cùng kiểu với nhau – kiểu cơ sở
- NNLT C luôn chỉ định một khối nhớ liên tục cho một biến kiểu mảng
- Kích thước của mảng được xác định ngay khi khai báo và không bao giờ thay đổi

Mảng – Khai báo trong C

```
typedef kiểu cơ sở Tênkiểu [Số thành phần];
```

kiểu của mỗi thành phần

hàng số, số thành phần
tối đa của mảng

do lập trình viên đặt tên

```
typedef int AINT [100];
```

//AINT là kiểu mảng biểu diễn dãy gồm 100 thành phần int

```
AINT a; //a: biến kiểu AINT
```

Mảng – Ví dụ

```
#define    SIZE        10

int        a[5];        // a dãy gồm 5 số nguyên
long int  big[100];    // big: chiếm 400 bytes!
double    d[100];     // d: chiếm 800 bytes!
long double v[SIZE];  // v:10 long doubles
```

```
long double d[2.5];
long double d[0];
long double d[-4];
long double d[n];
```

Mảng – Ví dụ

```
int    a[5]    = { 10, 20, 30, 40, 50 };
double d[100]  = { 1.5, 2.7 };
short  primes[] = { 1, 2, 3, 5, 7, 11, 13 };
long   b[50]   = { 0 };
```

khởi trị cho 5
thành phần

compiler xác định
kích thước gồm 7
thành phần

2 thành phần
đầu tiên được
khởi trị, phần
còn lại: 0

cách nhanh nhất để
khởi trị tất cả các
thành phần bằng 0

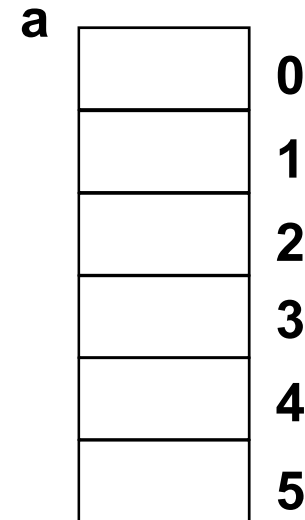
```
int    i = 7;
const int c = 5;
int    a[i];
double d[c];
short  primes[];
```



Mảng – Truy xuất các phần tử


- Các thành phần của mảng được truy xuất thông qua chỉ số của chúng 0..size-1
- Thao tác truy xuất không kiểm tra giới hạn của chỉ số nhưng giá trị không kiểm soát được

```
int main()
{
    int    a[6];
    int    i = 7;
    a[0] = 59;
    a[5] = -10;
    a[i/2] = 2;
    a[6] = 0;
    a[-1] = 5;
    return 0;
}
```



Truyền tham số Mảng cho hàm

- Tham số kiểu mảng được truyền cho hàm chính là địa chỉ của phần tử đầu tiên trên mảng
- Số thành phần trong tham số mảng có thể để trống.
- Số thành phần thực sự được sử dụng phải truyền qua một tham số khác (vd: size)



```
int add_elements(int a[], int size)
{
```

```
int add_elements(int *p, int size)
{
```

Ví dụ

```
#include <stdio.h>
void sum(long [], int);
int main(void) {
    long primes[6] = { 1, 2,
                      3, 5, 7, 11 };
    sum(primes, 6);
    printf("%li\n", primes[0]);
    return 0;
}
```

```
void sum(long a[], int sz) {
    int i;
    long total = 0;
    for(i = 0; i < sz; i++)
        total += a[i];
    a[0] = total;
}
```

primes

1
2
3
5
7
11

a

6

sz

dùng để kiểm tra
giới hạn chỉ số

tổng được lưu vào
phần tử đầu tiên

Chú ý

- Không thể thực hiện các thao tác chép nội dung một mảng sang mảng khác.

Chép từng phần tử mảng

```
char A[3]={'a','b','c'};
```

```
char B[3];
```

```
B = A; // ???
```

```
for(int i=0; i<3; i++)
```

```
    B[i] = A[i];
```

hoặc chép khối bộ nhớ (sẽ được đề cập sau)

- Không dùng phép so sánh trực tiếp (==) nội dung trong hai mảng.

Phép so sánh (A==B) so sánh địa chỉ hai vùng nhớ mà A và B chỉ đến.

Một số thao tác cơ sở


- Nhập
- Xuất
- Thêm một thành phần dữ liệu
- Loại bỏ một thành phần dữ liệu
- Tìm kiếm
- Sắp xếp

Mảng – Nhập dữ liệu

```
void ReadData (int a[], int size)
{
    int i;

    for (i = 0; i < size; i++)
    {
        printf("Nhap thanh phan %d: ", i);
        scanf("%d", &a[i]);
    }
}
```

duyệt qua tất cả các
phần tử



nhập dữ liệu cho a[i]



Mảng – Xuất dữ liệu ra màn hình

```
void WriteData(int a[], int size)
{
    int i;

    for(i = 0; i < size; i++)
        printf("%d ", a[i]);
    printf("\n");
}
```

Mảng – Nhập xuất dữ liệu

```
#include <stdio.h>
void ReadData(int [], int );
void WriteData(int [], int );
void main()
{
    int a[100], n;
    clrscr();
    printf("Nhap so thanh phan cua day: ");
    scanf("%d", &n);
    printf("Nhap cac thanh phan cua day: ");
    ReadData(a, n);
    printf("Day vua nhap: \n");
    WriteData(a, n);
}
```

Mảng – Tìm vị trí X trong dãy

- Bài toán: Tìm vị trí X trên mảng a đang có N thành phần.
- Giải pháp: Tìm tuần tự

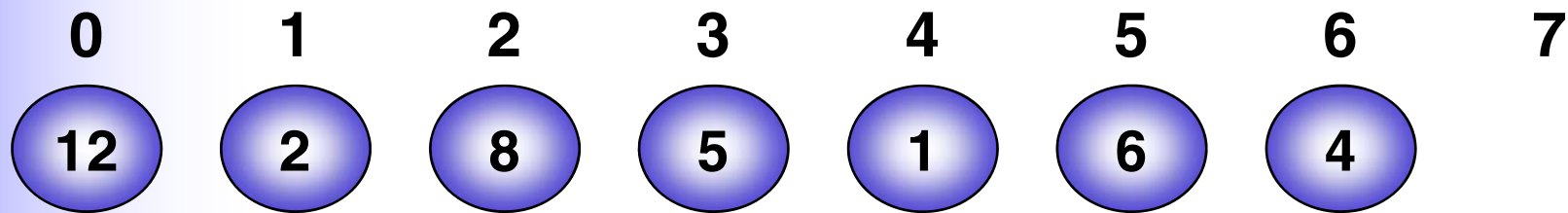
```
//input: dãy (a, N), X  
//output: Vị trí của X, -1 nếu không có  
  
int Search(int a[], int N, int X)  
{  
    for (int i = 0; i < N; i ++)  
        if (a[i] == X)  
            return i;  
    return -1;  
}
```


Mảng – Thêm một thành phần dữ liệu

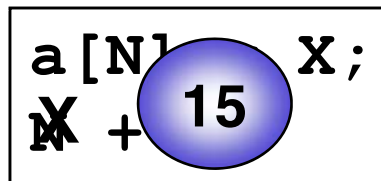
- Bài toán: cần thêm thành phần dữ liệu X vào mảng a đang có N thành phần.
- Hai trường hợp cần xem xét:
 - Dãy chưa có thứ tự
 - Thêm X vào cuối a .
 - Dãy đã có thứ tự
 - Tìm vị trí thích hợp, chèn X vào

Mảng – Thêm X vào cuối dãy

Thêm 15 vào (a, 7)

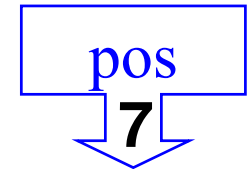
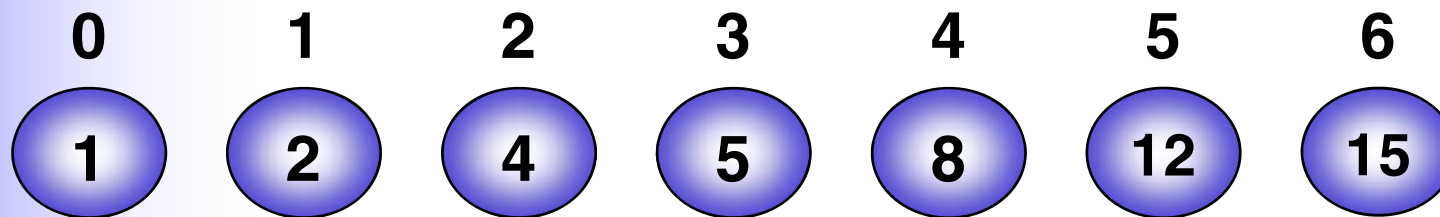


$N = 8$



Mảng – Chèn X vào dãy tăng dần

Chèn 6 vào (a, 7)



N = 8

X 6

Vị trí thích hợp: 4

Mảng – Chèn X vào dãy tăng dần

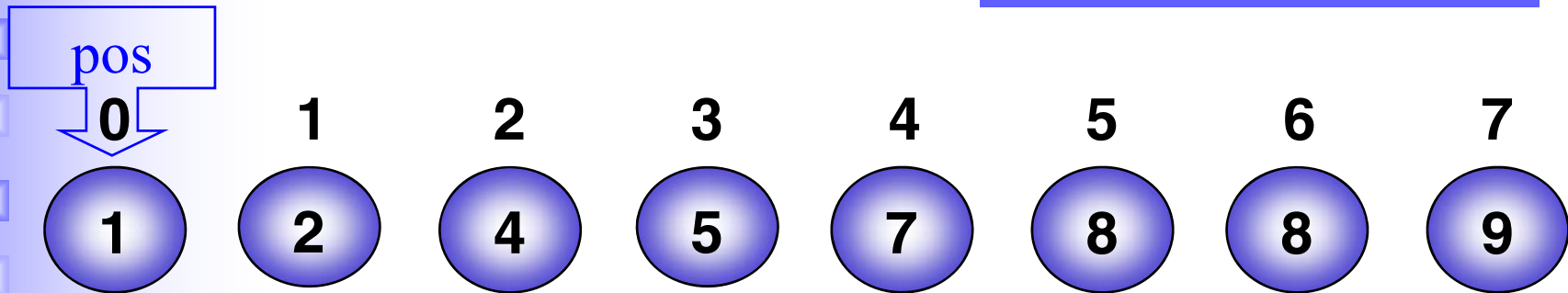
```
//input: dãy (a, N) tăng dần, X  
//output: dãy (a, N) đã có X ở đúng vị trí  
  
void    Insert(int a[], int &N, int X)  
{  
    int pos;  
  
    for (pos = N; (pos>0) && (a[pos-1]>X); pos --)  
        a[pos] = a[pos - 1];  
    a[pos] = X;  
    N ++;  
}
```

Mảng – Loại bỏ một thành phần dữ liệu

- Bài toán: loại bỏ thành phần dữ liệu X ra khỏi mảng a đang có N thành phần.
- Hướng giải quyết: xác định vị trí của X , nếu tìm thấy thì dồn các phần tử ở phía sau lên để lấp vào chỗ trống. 2 trường hợp:
 - Dãy không có thứ tự: lấp phần tử cuối lên
 - Dãy đã thứ tự: dời tất cả các phần tử ở sau vị trí của X lên trước 1 vị trí.

Mảng – Loại bỏ X ra khỏi dãy tăng

Loại 5 khỏi (a, 8)



N = 7

X 5



Ok, found

Tìm vị trí của 5

Đồn các vị trí 4, 5, 6, 7 lên

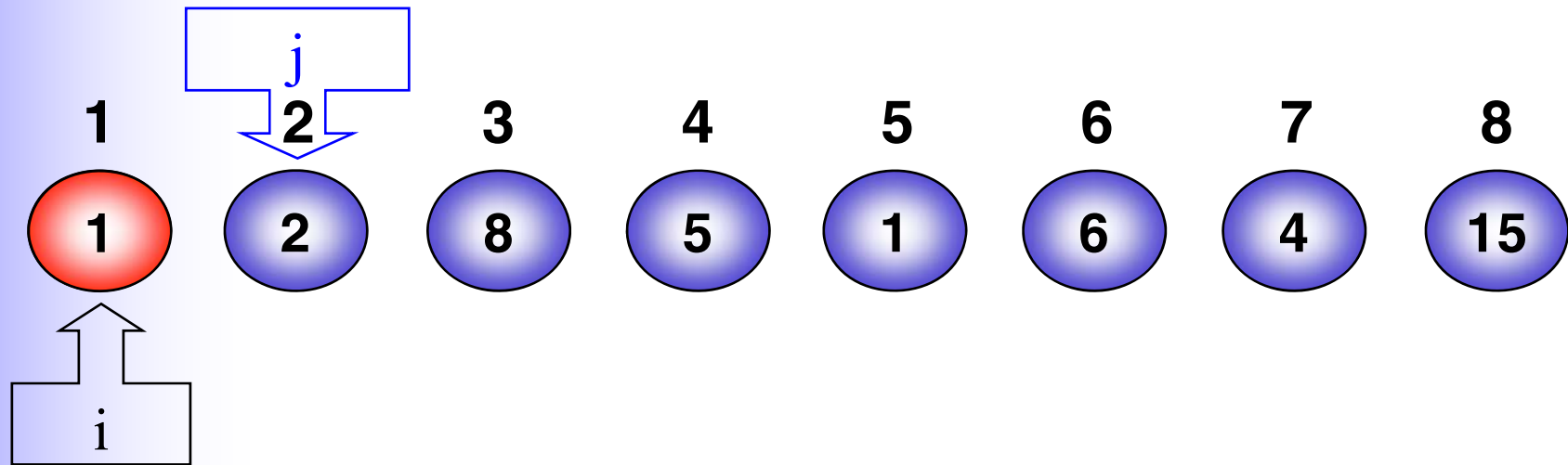
Mảng – Loại bỏ X ra khỏi dãy tăng

```
//input: dãy (a, N), X  
//output: dãy (a, N) đã loại bỏ 1 thành phần X  
  
int    Remove(int a[], int &N, int X)  
{  
    int pos = Search(a, N, X);  
    if (pos == -1) //không có X trong dãy  
        return 0;  
    N --;  
    for (; (pos < N); pos ++)  
        a[pos] = a[pos + 1];  
    return 1;  
}
```

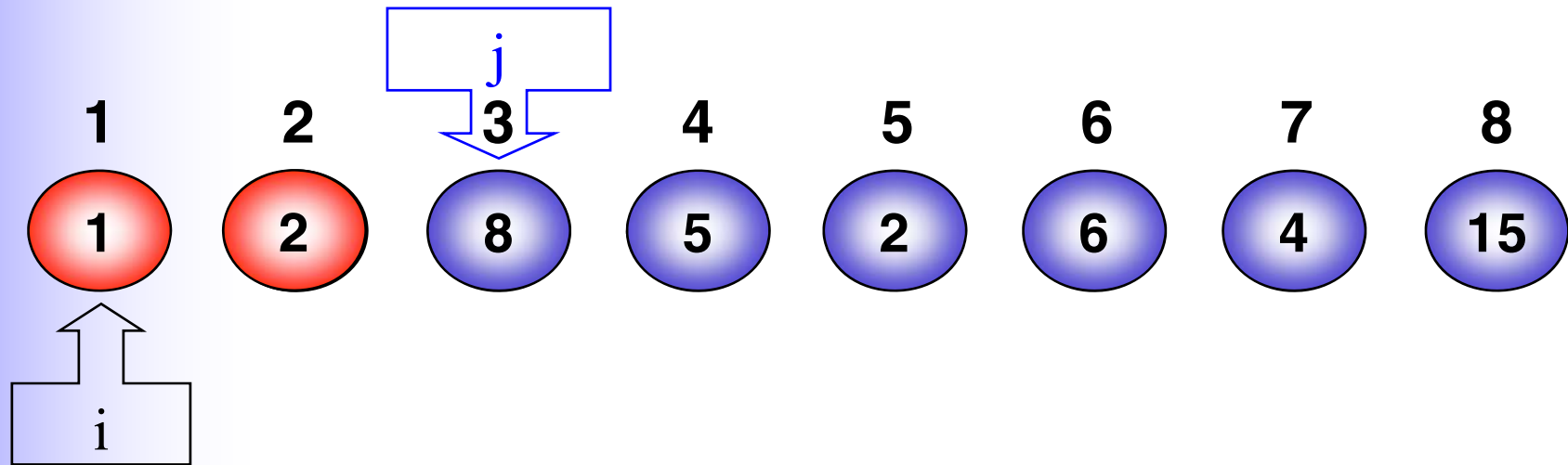
Mảng – Sắp xếp

- Bài toán: Sắp xếp các thành phần của (a, N) để thu được dãy tăng dần
- Giải pháp: Tìm cách triệt tiêu tất cả các nghịch thế của dãy
 - Thuật toán sắp xếp **Đổi chỗ trực tiếp**

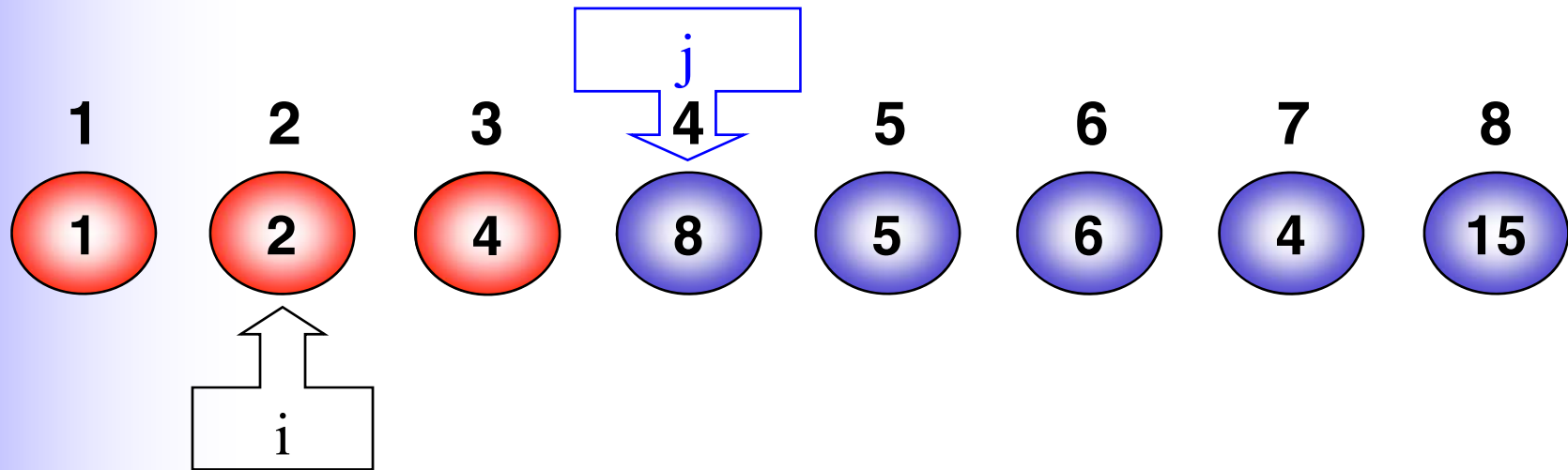
Mảng – Sắp xếp đôi chỗ



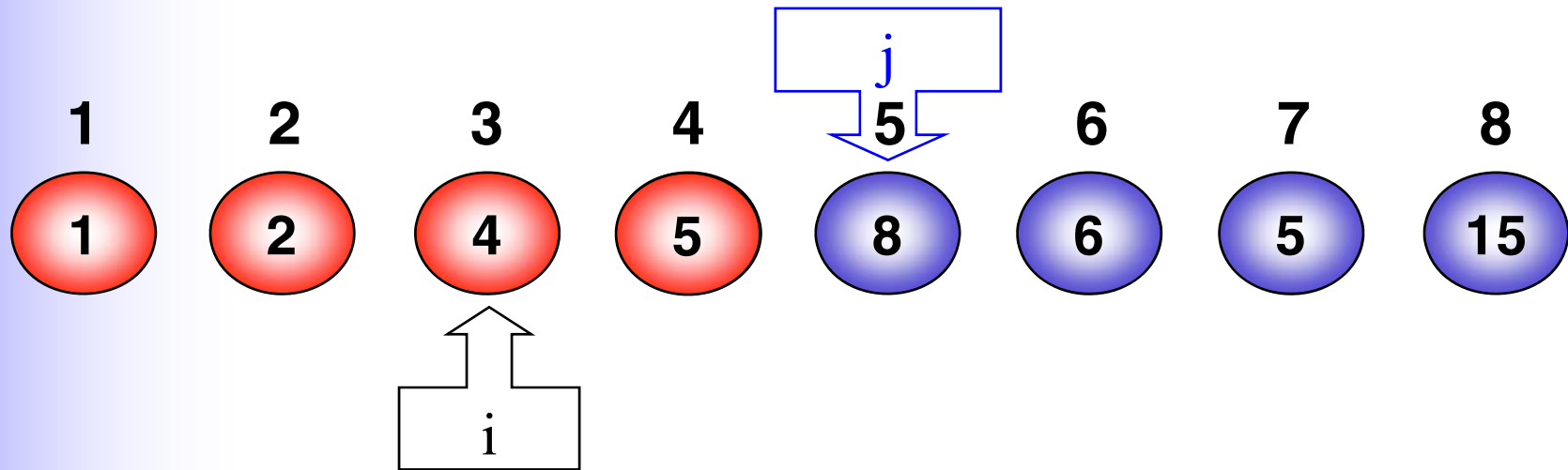
Mảng – Sắp xếp đổi chỗ



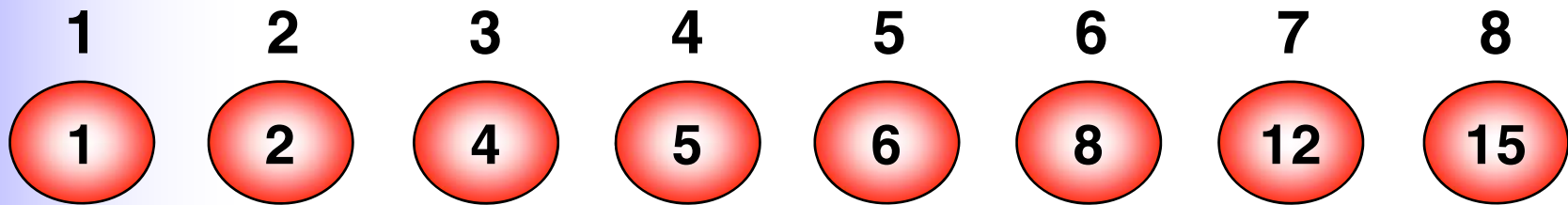
Mảng – Sắp xếp đổi chỗ



Mảng – Sắp xếp đổi chỗ



Mảng – Sắp xếp đổi chỗ



Mảng – Sắp xếp đổi chỗ

```
void Swap(int &x, int &y)
{
    int t = x; x = y; y = t;
}

void InterchangeSort(int a[], int N)
{
    int i, j;
    for (i = 0 ; i<N-1 ; i++)
        for (j =i+1; j < N ; j++)
            if(a[j]< a[i])
                Swap(a[i],a[j]);
}
```

Mảng nhiều chiều

- C không hỗ trợ mảng nhiều chiều. Tuy nhiên có thể tiếp cận theo hướng: Mảng 2 chiều là mảng một chiều mà mỗi thành phần của nó là một mảng một chiều.

```
float    rainfall[12][365];
```

“rainfall” là mảng gồm 12 thành phần, mỗi thành phần là mảng gồm 365 số float

```
short    exam_marks[500][10];
```

“exam_marks” là mảng gồm 500 thành phần, mỗi thành phần là mảng 10 số short

```
const int brighton = 7;
```

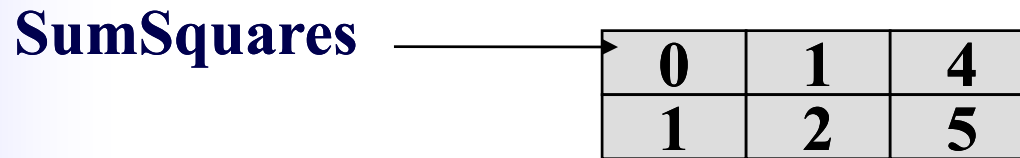
```
int day_of_year = 238;
```

```
rainfall[brighton][day_of_year] = 0.0F;
```

Mảng nhiều chiều

- Khai báo mảng 2 chiều:

```
type name[row_size][column_size];
```



0	1	4	1	2	5
---	---	---	---	---	---

- `int SumSquares[2][3] = { {0,1,4}, {1,2,5} };`
- `int SumSquares[2][3] = { 0,1,4,1,2,5 };`
- `int SumSquares[2][3] = { {0,1,4} };`
- `int SumSquares[][3] = { {0,1,4}, {1,2,5} };`
- `int SumSquares[][3] = { {0,1, }, {1} };`
- `int SumSquares[][3];`

Nhập Mảng 2 chiều

```
1. #define MAX_STUDENT 5
2. #define MAX_SUBJECT 6
3. int StudentScore[MAX_STUDENT][MAX_SUBJECT];
4. void read_Score(int Score[MAX_STUDENT][MAX_SUBJECT],
                   int nStudents, int nSubjects)
5. {
6.     int i,j;
7.     for(i=0; i<nStudents; i++)
8.         for(j=0; j<nSubjects; j++)
9.             scanf("%d", &Score[i][j]);
10. }
```

Hàm truy cập, in Mảng 2 chiều

```
1. void print_Score(int Score[MAX_STUDENT][MAX_SUBJECT],  
                    int nStudents, int nSubjects)  
2. {  
3.     int i,j;  
4.     for(i=0; i<nStudents; i++)  
5.     {  
6.         for(j=0; j<nSubjects; j++)  
7.             printf("%2d\t", &Score[i][j]);  
8.         printf("\n");  
9.     }  
10. }
```

Chương trình

```
1.  main(void)
2.  {
3.    int nStudents, nScores;
4.    scanf(“%d %d”, &nStudents, &nScores);
5.    if(nStudents <= MAX_STUDENT &&
        nScores <= MAX_SCORES)
6.        read_Score(StudentScore, nStudents, nScores);
7.    print_Score (StudentScore, nStudents, nScores);
8.    return 0;
9.  }
```

Biểu diễn mảng 2 chiều

StudentScores

Student1

Student2

Student3

Student4

Student5

0	1	4	?	?	?
1	2	5	?	?	?
?	?	?	?	?	?
?	?	?	?	?	?
?	?	?	?	?	?

0	1	4	?	?	?	1	2	5	?	?	?
---	---	---	---	---	---	---	---	---	---	---	---

Student1

Student2



Con trỏ - Pointer

Phạm Thế Bảo

Trường Đại học Khoa học Tự nhiên Tp.HCM

Con trỏ – Pointer

- Khai báo
- Các toán tử “&”, “*”, “=”, “+”
- Nhắc lại về truyền tham số địa chỉ
- Con trỏ và mảng
- Cấp phát vùng nhớ động

Con trỏ – Một số lý do nên sử dụng

- Con trỏ là kiểu dữ liệu lưu trữ địa chỉ của các vùng dữ liệu trong bộ nhớ máy tính
- Kiểu con trỏ cho phép:
 - Truyền tham số kiểu địa chỉ
 - Biểu diễn các kiểu, cấu trúc dữ liệu động
 - Lưu trữ dữ liệu trong vùng nhớ heap
- Con trỏ đã được sử dụng trong hàm `scanf`

Con trỏ – Khai báo trong C

Kiểu con trỏ phải được định nghĩa trên một kiểu cơ sở đã được định nghĩa trước đó.

```
typedef kiểucơsở *Tênkiểu;
```

```
typedef      int      *PINT;
```

//PINT là kiểu con trỏ - địa chỉ vùng nhớ kiểu int

```
int      x;
```

```
PINT     p; //p, p1: biến kiểu int *
```

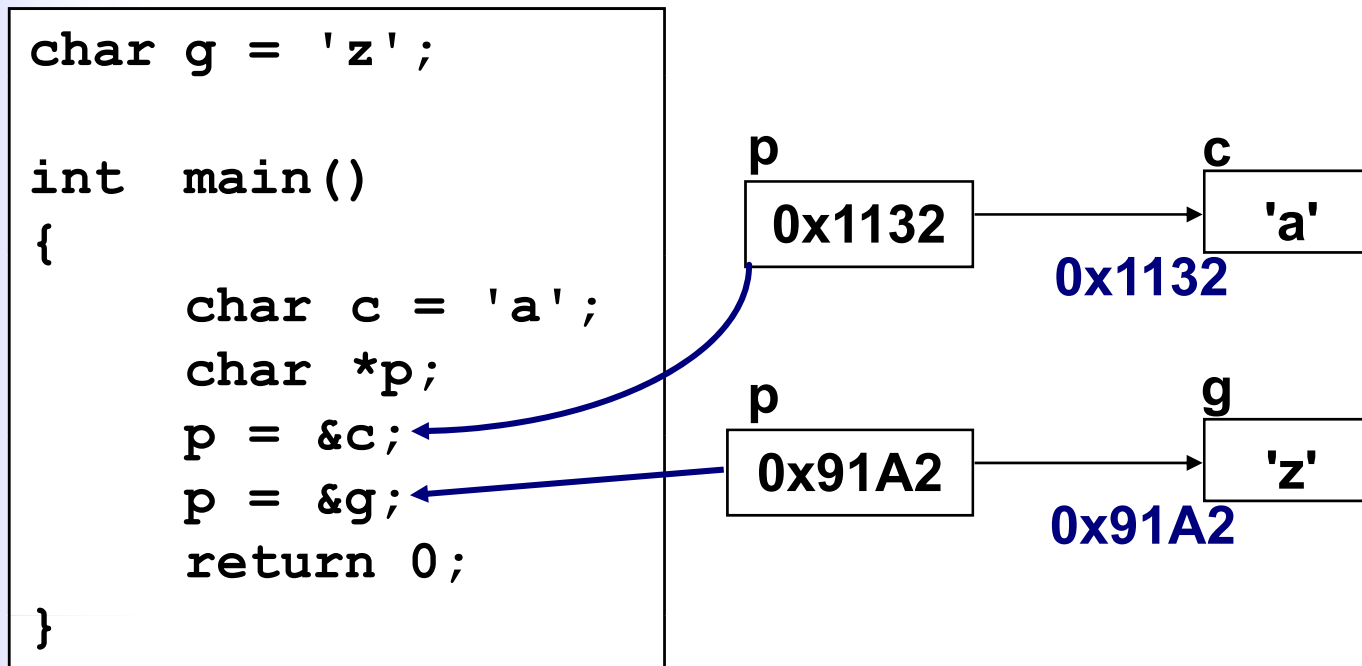
```
int      *p1;
```


Con trỏ – Khai báo trong C

```
int      *pi;  
  
long int *p;  
  
float*   pf;  
  
char     c, d, *pc; /* c và d kiểu char  
                  pc là con trỏ đến char */  
  
double*  pd, e, f; /* pd là con trỏ đến double  
                  e and f are double */  
  
char     *start, *end;
```

Con trỏ - Toán tử “&”

- “&”: toán tử lấy địa chỉ của 1 biến
- Địa chỉ của tất cả các biến trong chương trình đều đã được chỉ định từ khi khai báo

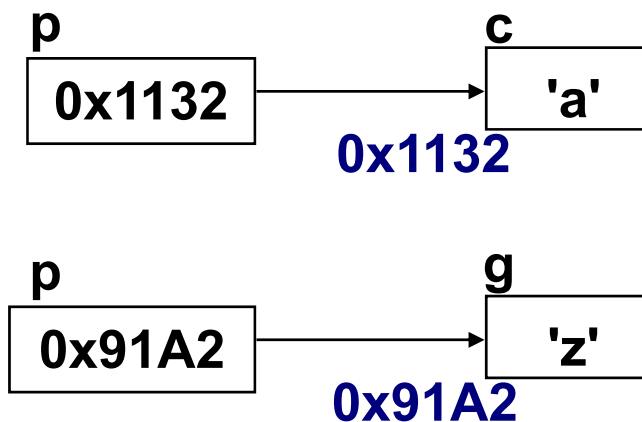


Con trỏ - Toán tử “*”

- “*”: toán tử truy xuất giá trị của vùng nhớ được quản lý bởi con trỏ.

```
#include <stdio.h>
char g = 'z';
int main()
{
    char c = 'a';
    char *p;
    p = &c;
    printf("%c\n", *p);
    p = &g;
    printf("%c\n", *p);
    return 0;
}
```

a
z



xuất giá trị do p đang quản lý

Con trỏ - Truyền tham số địa chỉ

```
#include <stdio.h>
void change(int *v);

int main()
{
    int var = 5;
    change(&var);
    printf("main: var = %i\n", var);
    return 0;
}

void change(int *v)
{
    (*v) *= 100;
    printf("change: *v = %i\n", (*v));
}
```

Con trỏ NULL

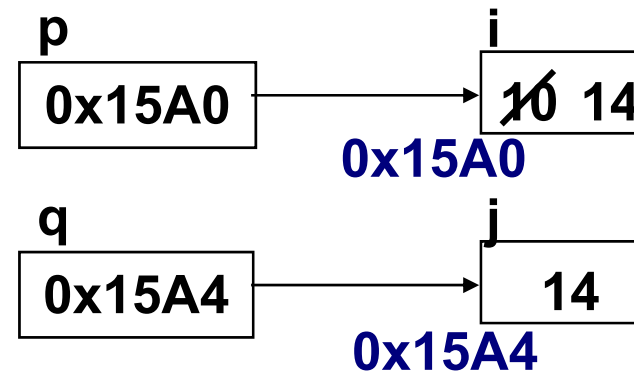
- Giá trị đặc biệt để chỉ rằng con trỏ không quản lý vùng nào. Giá trị này thường được dùng để chỉ một con trỏ không hợp lệ.

```
#include <stdio.h>
int main()
{
    int i = 13;
    short *p = NULL;
    if (p == NULL)
        printf("Con trỏ không hợp lệ!\n");
    else
        printf("Giá trị : %hi\n", *p);
    return 0;
}
```

Con trỏ - Toán tử gán “=”

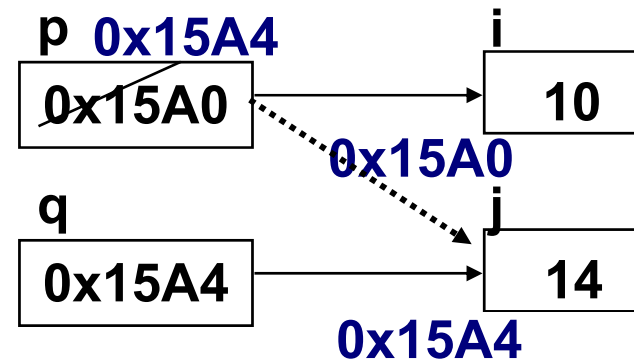
- Có sự khác biệt rất quan trọng khi thực hiện các phép gán:

```
int i = 10, j = 14;  
int* p = &i;  
int *q = &j;  
  
*p = *q;
```



và:

```
int i = 10, j = 14;  
int *p = &i;  
int *q = &j;  
  
p = q;
```



Luyện tập – Điền vào ô trống

```
int main(void)
{
    int i = 10, j = 14, k;
    int *p = &i;
    int *q = &j;

    *p += 1;
    p = &k;
    *p = *q;
    p = q;
    *p = *q;

    return 0;
}
```

i
0x2100

j
0x2104

k
0x1208

p
0x120B

q
0x1210

Con trỏ và Mảng

- Biến kiểu mảng là địa chỉ tĩnh của một vùng nhớ, được xác định khi khai báo, không thay đổi trong suốt chu kỳ sống.
- Biến con trỏ là địa chỉ động của một vùng nhớ, được xác định qua phép gán địa chỉ khi chương trình thực thi.

```
#include <stdio.h>
int main()
{
    int a[10] = {1, 3, 4, 2, 0};
    int *p;
    p = a; //a = p: sai
    printf("0x%04X %i 0x%04X %i\n",
           a, a[0], p, *p);
    return 0;
}
```


Con trỏ - Toán tử “+” với số nguyên

```
#include <stdio.h>
int main()
{
    short a[10] = {1, 3, 5, 2, 0};
    short *p = a;
    printf("0x%04X %i 0x%04X %i\n",
           a, a[0], p, *p);
    p ++;
    printf("0x%04X %i 0x%04X %i\n",
           a, a[0], p, *p);
    (*p) ++;
    printf("0x%04X %i 0x%04X %i\n",
           a, a[0], p, *p);
    return 0;
}
```

0x15A0

a

1
3
5
2
0
...

0x16B2

p

0x15A0

Con trỏ - Luyện tập

```
#include <stdio.h>
int main()
{
    int a[10] = {2, 3, 5, 1, 4, 7, 0};
    int *p = a;
    printf("%i %i\n", a[0], *p);
    p ++;
    printf("%i %i\n", *p, p[2]);
    p ++;  a[2] = 9;
    printf("%i %i\n", p[1], *p);
    p -= 2;
    printf("%i %i\n", p[3], p[1]);
    return 0;
}
```

2	2
3	1
1	9
1	3

Con trỏ - Cấp phát vùng nhớ động

- Có thể chỉ định vùng mới cho 1 con trỏ quản lý bằng các lệnh hàm **malloc**, **calloc** hoặc toán tử **new** của C++
- Vùng nhớ do lập trình viên chỉ định phải được giải phóng bằng lệnh **free** (malloc, calloc) hoặc toán tử **delete** (new)

```
#include <stdio.h>
int main()
{
    int *p = new int[10];
    p[0] = 1;
    p[3] = -7;
    delete []p;
    return 0;
}
```

Cấp phát động: malloc() và calloc()

- Hàm malloc và calloc cho phép cấp phát các vùng nhớ ngay trong lúc chạy chương trình.

```
void *malloc( size_t size);
```

```
void *calloc( size_t nItems, size_t size);
```

- Hàm calloc cấp phát vùng nhớ và khởi tạo tất cả các bit trong vùng nhớ mới cấp phát về 0.
- Hàm malloc chỉ cấp phát vùng nhớ.

Ví dụ 1: dùng malloc()

```
1. #include <stdio.h>
2. #include <string.h>
3. #include <alloc.h>
4. #include <process.h>

5. int main(void)
6. { char *str;
7.   /* allocate memory for string */
8.   if ((str = (char *) malloc(10)) == NULL)
9.     {
10.    printf("Not enough memory to allocate buffer\n");
11.    exit(1); /* terminate program if out of memory */
12.    }
```

Ví dụ 1: (tt)

```
13.     /* copy "Hello" into string */
14.     strcpy(str, "Hello");

15.     /* display string */
16.     printf("String is %s\n", str);

17.     /* free memory */
18.     free(str);
19.     return 0;
20. }
```

Ví dụ 2: calloc()

```
1. #include <stdio.h>
2. #include <alloc.h>
3. #include <string.h>

4. int main(void)
5. {
6.     char *str = NULL;

7.     /* allocate memory for string */
8.     str = (char *) calloc(10, sizeof(char));

9.     /* copy "Hello" into string */
10.    strcpy(str, "Hello");
```

Ví dụ 2: calloc()

```
11.    /* display string */
12.    printf("String is %s\n", str);

13.    /* free memory */
14.    free(str);

15.    return 0;
16. }
```


Giải phóng vùng nhớ

- Khi thoát khỏi hàm, các biến khai báo trong hàm sẽ “biến mất”. Tuy nhiên các vùng nhớ được cấp phát động vẫn còn tồn tại và được “đánh dấu” là đang “được dùng” → bộ nhớ của máy tính sẽ hết.

- ví dụ:

```
void aFunction(void)
```

```
{
```

```
    int *tmpArray, i = 5;
```

```
    tmpArray = (int *)malloc(i * sizeof(int));
```

```
}
```

tmpArray

i



Giải phóng vùng nhớ: free

- Sử dụng các cặp hàm

(malloc, free)

(calloc, free)

- C dùng hàm free để giải phóng vùng nhớ cấp phát động.

```
void free(void *block);
```

Cấp phát lại vùng nhớ: realloc

- Đôi khi chúng ta muốn mở rộng hoặc giảm bớt kích thước mảng.
- C dùng hàm realloc để cấp phát lại vùng nhớ, thực hiện chép nội dung của vùng nhớ cũ sang vùng nhớ mới.

```
void *realloc(void *block, size_t size);
```

- ví dụ:

```
int *tmpArray, N=5,i;
```

```
tmpArray = (int *)malloc(N * sizeof(int));
```

```
for(i = 0; i<N; i++)
```

```
    tmpArray[i] = i;
```

```
tmpArray = (int *)realloc(tmpArray, 7);
```



Ví dụ: realloc()

1. `#include <stdio.h>`
2. `#include <alloc.h>`
3. `#include <string.h>`
4. `int main(void)`
5. `{`
6. `char *str;`
7. `/* allocate memory for string */`
8. `str = (char *) malloc(10);`

Ví dụ: realloc() - tt

```
9.  /* copy "Hello" into string */
10. strcpy(str, "Hello");

11. printf("String is %s\n Address is %p\n", str, str);
12. str = (char *) realloc(str, 20);
13. printf("String is %s\n New address is %p\n", str, str);

14. /* free memory */
15. free(str);

16. return 0;
17. }
```

Di chuyển con trỏ trong mảng

```
int IntArr[5] = {3,4,2,1,0};
```

```
char CharArr[5] = {'a','b','c','d','e'};
```

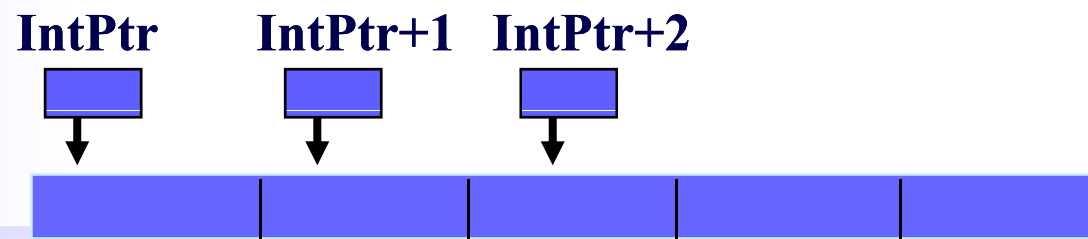
```
int *IntPtr = IntArr;
```

```
char *charPtr = CharArr;
```

- Di chuyển trong mảng 1 chiều:

```
int val_at_0 = * IntPtr;    // = IntArr[0]
```

```
int val_at_3 = * (IntPtr+3); // = IntArr[3]
```



Phép toán với con trỏ trong mảng

```
#define ROWS    5
```

```
#define COLS    3
```

```
int IntArr[ROWS][COLS] = {{3,4,2,1,0},  
                           {5,6,7,8,9},  
                           {10,11,12,13,14}};
```

```
int *IntPtr = IntArr;
```

- Tham chiếu đến phần tử trong mảng 2 chiều:

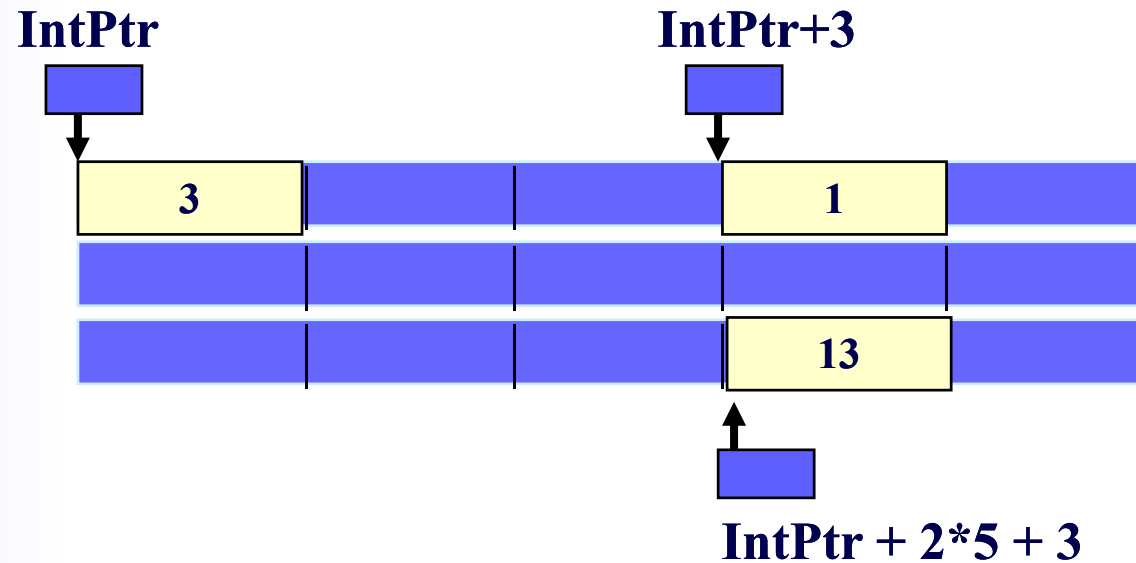
```
int val_at_00 = * IntPtr;// = IntArr[0][0]
```

```
int val_at_30 = * (IntPtr+3);// = IntArr[3][0]
```

```
int val_at_32 = * (IntPtr+3*5+2);// = IntArr[3][2]
```

```
int val_at_32 = * ( * (IntPtr+3)
```

Phép toán với con trỏ trong mảng





Ký tự và Chuỗi ký tự - String

Phạm Thế Bảo

Trường Đại học Khoa học Tự nhiên Tp.HCM

Ký tự (character)

- Kiểu char:
 - ký tự “in được” gồm: 26 chữ thường (a..z), 26 chữ hoa (A..Z), 10 chữ số (0..9), khoảng trắng, các ký tự:
! “ # \$ % & ‘ () * + , - . / : ; < = > ? @ [\] ^ _ { | } ~
 - Các ký tự “không in được”: tab, lert (bell), newline, formfeed,...
- các ký tự “in được” đặc biệt: “\\”, “\”, “\””
- các ký tự “không in được” đặc biệt:
 - \n new line
 - \a bell
 - \0 null character
 - \b backspace
 - \t horizontal tab
 - ...

Nhập xuất Ký tự

- scanf

```
char ch;  
scanf("%c", &ch);
```

- sử dụng các đoạn macro có trong thư viện <stdio.h>

putchar: đưa ký tự ra thiết bị xuất chuẩn (stdout)

```
putchar('\n');
```

getchar: lấy ký tự từ thiết bị nhập chuẩn (stdin)

```
ch = getchar();
```

- getch: lấy trực tiếp ký tự từ bàn phím không hiển thị ra màn hình

```
ch = getch();
```

getche(): lấy trực tiếp ký tự từ bàn phím và hiển thị ký tự ra màn hình.

```
ch = getche();
```

getchar

```
1. #include <stdio.h>
2. int main(void)
3. {
4.     int c;
5.     /* Note that getchar reads from stdin and is line buffered; this means it
6.     will not return until you press ENTER. */
7.     while ((c = getchar()) != '\n')
8.         printf("%c", c);
9.     return 0;
}
```

putchar

```
1.  /* putchar example */
2.  #include <stdio.h>

3.  /* define some box-drawing characters */
4.  #define LEFT_TOP 0xDA
5.  #define RIGHT_TOP 0xBF
6.  #define HORIZ   0xC4
7.  #define VERT    0xB3
8.  #define LEFT_BOT 0xC0
9.  #define RIGHT_BOT 0xD9

10. int main(void)
11. {
12.     char i, j;
13.     /* draw the top of the box */
14.     putchar(LEFT_TOP);
15.         for (i=0; i<10; i++)
16.             putchar(HORIZ);
```

```
17.     putchar(RIGHT_TOP);
18.     putchar('\n');

19.  /* draw the middle */
20.  for (i=0; i<4; i++)
21.  {
22.      putchar(VERT);
23.      for (j=0; j<10; j++)
24.          putchar(' ');
25.      putchar(VERT);

26.      putchar('\n');
27.  }
28.  /* draw the bottom */
29.  putchar(LEFT_BOT);
30.  for (i=0; i<10; i++)
31.      putchar(HORIZ);
32.  putchar(RIGHT_BOT);
33.  putchar('\n');

34.  return 0;
35. }
```

Một số hàm khác

- kbhit: kiểm tra có phím bấm

1. `/* khit example */`
2. `#include <conio.h>`
3. `int main(void)`
4. `{`
5. `cprintf("Press any key to continue:");`
6. `while (!kbhit()) /* do nothing */ ;`
7. `cprintf("\r\nA key was pressed...\r\n");`
8. `return 0;`
9. `}`

Chuỗi ký tự – Strings

- Một số qui tắc
- Nhập / xuất
- Con trỏ và chuỗi ký tự
- Một số hàm thư viện

Chuỗi ký tự - Một số quy tắc

- Chuỗi ký tự là mảng một chiều có mỗi thành phần là một số nguyên được kết thúc bởi số 0.
- Ký tự kết thúc (0) ở cuối chuỗi ký tự thường được gọi là ký tự null (không giống con trỏ NULL). Có thể ghi là 0 hoặc '\0' (không phải chữ o).
- Được khai báo và truyền tham số như mảng một chiều.

```
char          s[100];  
unsigned char s1[1000];
```

Chuỗi ký tự - Ví dụ

```
char first_name[5] = { 'J', 'o', 'h', 'n', '\0' };  
char last_name[6]  = "Minor";  
char other[]       = "Tony Blurt";  
char characters[8] = "No null";
```

first_name	'J'	'o'	'h'	'n'	0						
last_name	'M'	'i'	'n'	'o'	'r'	0					
other	'T'	'o'	'n'	'y'	32	'B'	'l'	'u'	'r'	't'	0
characters	'N'	'o'	32	'n'	'u'	'l'	'l'	0			

Chuỗi ký tự - Nhập / xuất

- Có thể nhập / xuất chuỗi ký tự s bằng cách nhập từng ký tự của s
- Hoặc sử dụng các hàm `scanf` và `printf` với ký tự định dạng "%s"

```
char other[] = "Tony Blurt";  
printf("%s\n", other);
```

- Nhập chuỗi có khoảng trắng dùng hàm `gets`

```
char name[100];  
printf("Nhap mot chuoai ky tu %s: ");  
gets(name);
```

Lưu ý: kết thúc chuỗi

```
#include <stdio.h>

int main()
{
    char other[] = "Tony Blurt";

    printf("%s\n", other);

    other[4] = '\\0';

    printf("%s\n", other);

    return 0;
}
```

"Blurt" sẽ không
được in ra

Tony Blurt
Tony

other

'T'	'o'	'n'	'y'	32	'B'	'l'	'u'	'r'	't'	0
-----	-----	-----	-----	----	-----	-----	-----	-----	-----	---

Lỗi khi tạo một chuỗi

- Chú ý: không có phép gán trong kiểu dữ liệu chuỗi
- như thế này là sai

```
char ten[10];  
ten = "hoahong";
```

Chú ý

- Không :
sử dụng toán tử gán = để chép nội dung của một chuỗi sang chuỗi khác.

```
char a[4]="hi";  
char b[4];  
b = a; //???
```

- Không:
dùng toán tử == để so sánh nội dung hai chuỗi

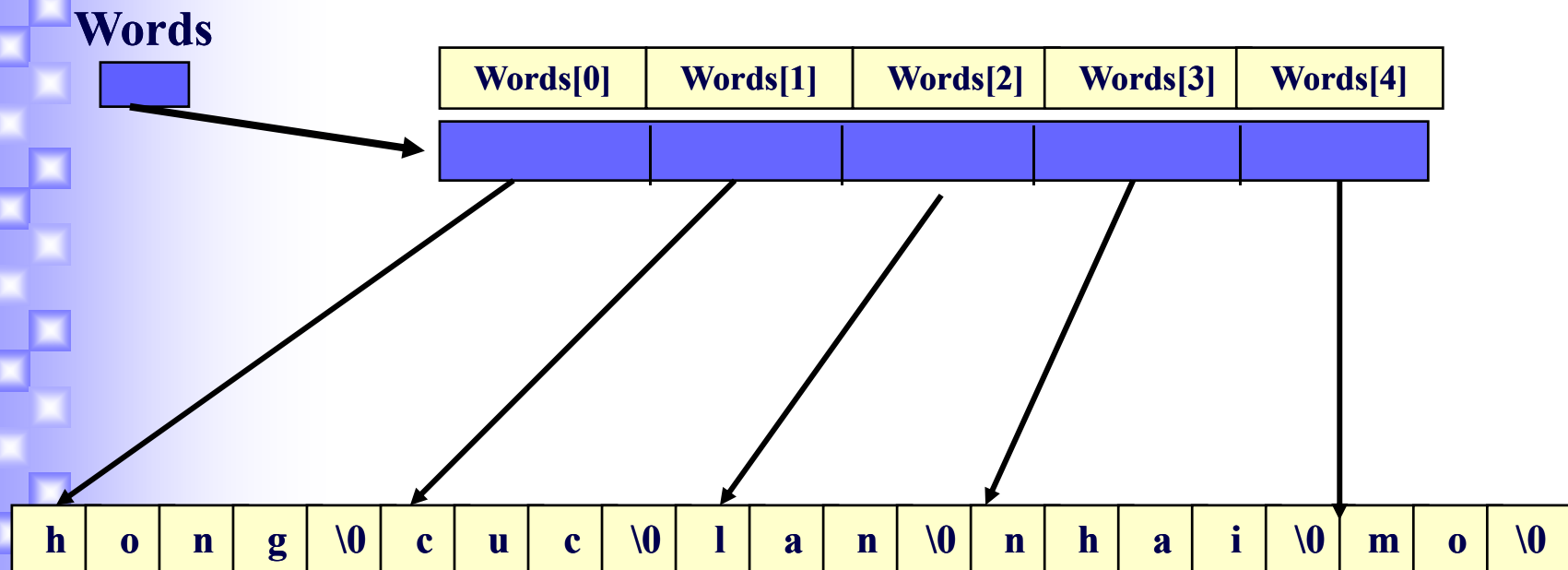
```
char a[] = "hi";  
char b[] = "there";  
if(a==b) //???  
{ }
```

Con trỏ và chuỗi ký tự

- `char *p;` // khai báo chuỗi ký tự như một con trỏ
- `p = new char[30];` //xin cấp phát số lượng ký tự,
// giống mảng các ký tự

Mảng các chuỗi: char * []

■ char *words[] = {"hong", "cuc", "lan", "nhai", "mo"};



Hay char **words;

Chuỗi ký tự – Một số hàm thư viện

- Lấy độ dài chuỗi

`l = strlen(s);`

- Đổi toàn bộ các ký tự của chuỗi thành IN HOA

`strupr(s);`

- Đổi toàn bộ các ký tự của chuỗi thành in thường

`strlwr(s);`

Chuỗi ký tự – Một số hàm thư viện

- So sánh chuỗi: so sánh theo thứ tự từ điển

Phân biệt IN HOA – in thường:

```
int strcmp(const char *s1, const char *s2);
```

Không phân biệt IN HOA – in thường:

```
int stricmp(const char *s1, const char *s2);
```

Chuỗi ký tự – ví dụ strcmp

```
#include <stdio.h>
#include <string.h>

int main()
{
    char s1[] = "Minor";
    char s2[] = "Tony";
    int cmp = strcmp(s1, s2);
    if (cmp < 0)
        printf("%s < %s", s1, s2);
    else
        if (cmp == 0)
            printf("%s = %s", s1, s2);
        else
            printf("%s > %s", s1, s2);
    return 0;
}
```

Minor < Tony

Chuỗi ký tự – Một số hàm thư viện

- Gán nội dung chuỗi:

- *Chép toàn bộ chuỗi source sang chuỗi dest:*

```
int strcpy(char *dest, const char *src);
```

- *Chép tối đa n ký tự từ source sang dest:*

```
int strncpy(char *dest,  
             const char *src, int n);
```

- Tạo chuỗi mới từ chuỗi đã có:

```
char *strdup(const char *src);
```

Chuỗi ký tự – ví dụ strcpy

```
#include <stdio.h>
#include <string.h>

int main()
{
    char s[] = "Tony Blurt";
    char s2[100], *s3;

    strcpy(s2, s);
    printf("%s\n", s2);
    strncpy(s2 + 2, "12345", 3);
    printf("%s\n", s2);
    s3 = strdup(s + 5);
    printf("%s\n", s3);
    free(s3);
    return 0;
}
```

```
Tony Blurt
To123Blurt
Blurt
```

Chuỗi ký tự – Một số hàm thư viện

- Nối chuỗi:

```
char *strcat(char *dest,  
             const char *src);
```

- Tách chuỗi:

```
char *strtok(char *s,  
             const char *sep);
```

Trả về địa chỉ của đoạn đầu tiên. Muốn tách đoạn kế tiếp tham số thứ nhất sẽ là NULL

Chuỗi ký tự – ví dụ strtok

```
#include <stdio.h>
#include <string.h>

#define SEPARATOR "., "

int main()
{
    char s[] = "Thu strtok: 9,123.45";
    char *p;

    p = strtok(s, SEPARATOR);
    while (p != NULL) {
        printf("%s\n", p);
        p = strtok(NULL, SEPARATOR);
    }
    return 0;
}
```

```
Thu
strtok:
9
123
45
```

Chuỗi ký tự – Một số hàm thư viện

- Tìm một ký tự trên chuỗi:

```
char *strchr(const char *s, int c);
```

- Tìm một đoạn ký tự trên chuỗi:

```
char *strstr(const char *s1,  
              const char *s2);
```


Chuỗi ký tự – ví dụ tìm kiếm

```
#include <stdio.h>
#include <string.h>

int main()
{
    char s[] = "Thu tim kiem chuoï";
    char *p;

    p = strchr(s, 'm');
    printf("%s\n", p);
    p = strstr(s, "em");
    printf("%s\n", p);
    return 0;
}
```

m kiem chuoï
em chuoï

Chuỗi ký tự – chèn một đoạn ký tự

```
#include <stdio.h>
#include <string.h>

void StrIns(char *s, char *sub)
{
    int len = strlen(sub);
    memmove(s + len, s, strlen(s)+1);
    strncpy(s, sub, len);
}

int main()
{
    char s[] = " Thu chen";

    StrIns(s, "123");          printf("%s\n", s);
    StrIns(s + 8, "45");      printf("%s\n", p);
    return 0;
}
```

123 Thu chen

123 Thu 45chen

Chuỗi ký tự – xóa một đoạn ký tự

```
#include <stdio.h>
#include <string.h>

void StrDel(char *s, int n)
{
    memmove(s, s + n, strlen(s+n)+1);
}
int main()
{
    char s[] = "Thu xoa 12345";

    StrDel(s, 4);          printf("%s\n", s);
    StrDel(s + 4, 3);     printf("%s\n", s);
    return 0;
}
```

xoa 12345
xoa 45

Biến đổi chuỗi sang số

- atoi(), atof(), atol():
đổi chuỗi ký tự sang số.

```
int atoi(const char *s);  
double atof(const char *s);  
long atol(const char *s);
```

- ...

```
float f;  
char *str = "12345.67";  
  
f = atof(str);  
printf("string = %s float = %f\n", str, f);
```

...



Cấu trúc - Struct

Phạm Thế Bảo

Trường Đại học Khoa học Tự nhiên Tp.HCM

Kiểu cấu trúc

- Khái niệm
- Khai báo
- Truy xuất các thành phần
- Cấu trúc & mảng
- Con trỏ đến cấu trúc

Khái niệm

- Cấu trúc là kiểu dữ liệu gồm một nhóm các thành phần có kiểu không giống nhau, mỗi thành phần được xác định bằng một tên riêng biệt (kiểu dữ liệu trừu tượng – Abstract Data Type – ADT).
- Kiểu của mỗi thành phần trong cấu trúc là một kiểu đã được định nghĩa trước, kể cả mảng và các cấu trúc khác.

Cấu trúc – Khai báo trong C

Một kiểu cấu trúc được định nghĩa với từ khóa **struct**.

```
typedef struct Tênkiểu
```

```
{
```

```
    Kiểuthànhphần        Tênthànhphần;
```

```
    Kiểuthànhphần        Tênthànhphần;
```

```
    Kiểuthànhphần        Tênthànhphần;
```

```
    Kiểuthànhphần        Tênthànhphần;
```

```
    . . .
```

```
};
```


Cấu trúc – ví dụ

```
typedef struct TDate
{
    char    day;
    char    month;
    int     year;
};
```

```
typedef struct TStudent
{
    char    ID[10];
    char    firstname[10];
    char    lastname[20];
    TDate   dob;
    float   marks[10];
};
```

```
typedef struct TBook
{
    char    title[80];
    char    author[80];
    float   price;
    char    isbn[20];
};
```

```
//khai báo các biến
TBook     book;
TStudent   list[100];
```

Cấu trúc – Truy xuất các thành phần

- Các thành phần của một biến kiểu cấu trúc được truy xuất thông qua tên biến, dấu "." và tên thành phần.

```
void Print(TStudent m)
{
    printf("Name           : %s %s\n",
           m.firstname, m.lastname);
    printf("Student ID      : %s\n", m.ID);
    printf("Date of birth   : %hi/%hi/%i",
           m.dob.day, m.dob.month, m.dob.year);
    printf("Marks           : ");
    for (int i=0; i<10; i++)
        printf("%.2f ", m.marks[i]);
}
```

Cấu trúc – Truy xuất các thành phần

```
void ReadInfo (TStudent &m)
{
    printf("Type student ID: ");
    scanf("%s", m.ID);
    printf("Type first name: ");
    gets(m.firstname);
    printf("Type last name: ");
    gets(m.lastname);
    printf("Date of birth (d m y): ");
    scanf("%hi %hi %i", &(m.dob.day),
          &(m.dob.month), &(m.dob.year));
    printf("Marks (10 floats): ");
    for (int i=0; i<10; i++)
        scanf("%f", &(m.marks[i]));
}
```

Kích thước của một cấu trúc

- Kích thước kiểu dữ liệu cấu trúc:

```
sizeof(<<typename>>)
```

- vd:

```
SinhVien_t_size = sizeof(SinhVien); // 48
```

Mảng các Cấu trúc

- Khai báo biến mảng các Cấu trúc cũng giống như khai báo biến mảng trên các kiểu dữ liệu cơ bản khác.
- Dùng tên mảng khi truy cập từng phần tử trong mảng các cấu trúc và toán tử thành viên để truy cập các trường dữ liệu của từng thành phần cấu trúc.

```
struct SinhVien {  
    char    hoten[31];  
    int     namsinh;  
    char    noisinh[4];  
    char    maso[11];  
}
```

```
SinhVien mang[100];  
    ...  
mang[i].namsinh
```

Con trỏ đến Cấu trúc

- Khai báo con trỏ đến cấu trúc tương tự như các kiểu dữ liệu khác.

```
SinhVien *SV_ptr
```

- Toán tử truy cập trường thành phần của cấu trúc do con trỏ chỉ đến: ->

```
SV_ptr->namsinh
```

```
SV_ptr->noisinh
```

Union

- Khai báo union được dùng để khai báo các biến dùng chung bộ nhớ.

```
union int_or_long {  
    int    i;  
    long   l;  
} a_number;
```

- Các thành phần của union có thể không có kích thước bằng nhau.
- Kích thước bộ nhớ trong khai báo union là kích thước của kiểu dữ liệu lớn nhất có trong khai báo union.

Ví dụ

```
1. union int_or_long {
2.     int    i;
3.     long   l;
4. } a_number;
5. a_number.i = 5;
6. a_number.l = 100L;

7. // anonymous union
8. union {
9.     int    i;
10.    float   f;
11. };
12. i = 10;
13. f = 2.2;
```


Kiểu Enum (liệt kê)

- Kiểu Enum:

- Liệt kê toàn bộ giá trị mà biến có thể nhận

- Khai báo:

```
enum <tên_kiểu_enum> {  
    <danh_sách_các_trị>  
};
```

- Ví dụ:

```
enum day {Hai, Ba, Tu, Nam, Sau, Bay, CN};  
    day ngay;  
    ngay = Tu; // 2  
  
enum modes {LASTMODE = -1, BW40 = 0}
```



Tập tin - File

Phạm Thế Bảo
Trường Đại học Khoa học Tự nhiên Tp.HCM

Kiểu cấu trúc

- Khái niệm
- Khai báo
- Phân loại
- Các thao tác

Khái niệm

- Tập tin là kiểu dữ liệu hỗ trợ truy xuất/lưu trữ dữ liệu trên bộ nhớ ngoài.
- C hỗ trợ hai kiểu tập tin: văn bản và nhị phân.
- Truy xuất thông qua con trỏ FILE.

Kiểu FILE *

- Kiểu FILE * (khai báo trong `stdio.h`) cho phép làm việc với các tập tin (văn bản, nhị phân).
- Khai báo con trỏ tập tin

```
FILE * fp;
```

- Chúng ta sử dụng con trỏ tập tin để truy cập (đọc, ghi, thêm thông tin) các tập tin.

Mở tập tin

```
FILE * fopen( const char *FileName, const char *Mode);
```

- Filename: tên tập tin cần mở. Có thể chỉ định một đường dẫn đầy đủ chỉ đến vị trí của tập tin.
- Mode: chế độ mở tập tin: chỉ đọc, để ghi (tạo mới), ghi thêm.
- Nếu thao tác mở thành công, fopen trả về con trỏ FILE trỏ đến tập tin FileName.
- Nếu mở không thành công (FileName không tồn tại, không thể tạo mới), fopen trả về giá trị NULL.

Đóng tập tin

```
int fclose( FILE *filestream );
```

- filestream: con trỏ đến tập tin đang mở cần đóng.
- Nếu thao tác đóng thành công, fclose trả về 0.
- Nếu có lỗi (tập tin đang sử dụng), fclose trả về giá trị EOF.

Ví dụ : Mở, Đóng tập tin

```
1. FILE * fp;  
  
2. // mở VB.TXT “chỉ đọc”  
3. if( (fp = fopen( “C:\\LTC\\VB.TXT”, “r” )) == NULL)  
4.     { printf( “Tap tin khong mo duoc\n”);  
5.       exit(1);  
6.     }  
  
    /* ... */  
  
7. fclose( fp );
```


Tập tin văn bản

- Tập tin văn bản là kiểu tập tin được lưu trữ các thông tin dưới dạng kiểu ký tự.
- Truy xuất tập tin văn bản:
 - theo từng ký tự
 - theo từng dòng
- Chế độ mở trên tập tin văn bản
 - “r” : đọc (tập tin phải có trên đĩa)
 - “w” : ghi (ghi đè lên tập tin cũ hoặc tạo mới nếu tập tin không có trên đĩa)
 - “a” : ghi nối vào cuối tập tin.
 - “r+” : đọc/ghi. Tập tin phải có trên đĩa.
 - “a+” : đọc, ghi vào cuối tập tin. Tạo mới tập tin nếu tập tin chưa có trên đĩa.

getc, putc, fgetc, fputc

- getc: nhận ký tự từ tập tin.

```
int getc ( FILE *fp );
```

getc trả về ký tự đọc được hoặc trả về EOF nếu fp không hợp lệ hoặc đọc đến cuối tập tin.

- putc: ghi ký tự ra tập tin.

```
int putc ( int Ch, FILE * fp );
```

putc trả về EOF nếu thao tác ghi có lỗi.

- có thể dùng fgetc và fputc.

Ví dụ 1: `getc()` đọc phím đến khi Enter

```
1. #include <stdio.h>
2. #include <conio.h>
3. #include <stdlib.h>
4. void main()
5.     { FILE * fp;
6.     char filename[67], ch;
7.     printf ( "Filename: " );
8.     gets (filename);
9.     if ((fp = fopen (filename, "w" )) == NULL ) // mở tập tin mới để ghi
10.        { printf ( "Create file error \n"); exit (1); }
11.     while (( ch = getche() ) != '\r' ) // đọc cho đến khi gặp ENTER
12.         putc ( fp );
13.     fclose ( fp );
14. }
```

Ví dụ 2: putc() in nội dung tập tin văn bản ra màn hình

```
1. #include <stdio.h>
2. #include <conio.h>
3. #include <stdlib.h>
4. void main()
5.     { FILE * fp;
6.     char filename[67], char ch;
7.     printf ( "Filename: " );
8.     gets (filename);
9.     if ((fp = fopen (filename, "r" )) == NULL ) // mở tập tin mới để đọc
10.        { printf ( "Open file error \n"); exit (1); }
11.     while (( ch = getc ( fp ) ) != EOF ) // đọc cho đến hết tập tin
12.        printf ( "%c", ch );
13.     fclose ( fp );
14. }
```

Ví dụ 3: Đếm số từ trong tập tin văn bản.

```
1. #include <stdio.h>
2. #include <conio.h>
3. #include <stdlib.h>
4. void main()
5.     { FILE * fp;
6.       char filename[67], char ch;
7.       int count = 0, isword = 0;

8.       printf ( "Filename: " ); gets (filename);
9.       if ((fp = fopen (filename, "r" )) == NULL ) // mở tập tin mới để đọc
10.          { printf ( "Open file error \n"); exit (1); }
```

```
11. while (( ch = getc ( fp ) ) != EOF ) // đọc cho đến hết tập tin
12.     {
13.         if (( ch >= 'a' && ch <= 'z' ) || ( ch >= 'A' && ch <= 'Z' ))
14.             isword =1;
15.         if (( ch == ' ' || ch == '\n' || ch == '\t' ) && isword )
16.             { count ++; isword = 0; }
17.     }

18.     printf ( "Number of word: %d\n", count);
19.     fclose ( fp );
20. }
```

fgets()

- fgets: đọc chuỗi ký tự từ tập tin.

```
char * fgets ( char *Str, int NumOfChar, FILE *fp );
```

fgets đọc các ký tự trong tập tin cho đến khi gặp một trong các điều kiện:

- EOF
 - gặp dòng mới
 - đọc được (NumOfChar - 1) ký tự trước khi gặp hai điều kiện trên.
- fgets trả về chuỗi ký tự đọc được (kết thúc bằng \0) hoặc trả về con trỏ NULL nếu EOF hoặc có lỗi khi đọc.

fputs()

- fputs: ghi chuỗi ký tự ra tập tin.

```
int fputs ( const char *Str, FILE * fp );
```

- fputs trả về EOF nếu thao tác ghi có lỗi.

Hàm chép tập tin văn bản

- Có trong các thư viện

```
#include <stdio.h>
```

```
#include <string.h>
```

- /* Viết chương trình chép từ SourceFile sang DestFile và trả về số ký tự đọc được

feof(); fscanf(), fprintf(); fflush()

- feof: cho biết đã đến cuối tập tin chữ a(EOF).
`int feof (FILE *fp);`
- fprintf:
`int fprintf (FILE *fp, const char * Format, ...);`
- fscanf:
`int fscanf (FILE *fp, const char * Format, ...);`
- fflush: làm sạch vùng đệm của tập tin
`int fflush (FILE * fp);`

Tập tin Nhị phân

- Tập tin nhị phân là một chuỗi các ký tự, không phân biệt ký tự in được hay không in được.
- Tập tin nhị phân thường dùng để lưu trữ các cấu trúc (struct) hoặc union.

- Khai báo:

```
FILE * fp;
```

- Truy xuất tập tin nhị phân theo khối dữ liệu nhị phân.

Tập tin Nhị phân

- Các chế độ mở tập tin nhị phân:
 - “rb” : mở chỉ đọc
 - “wb” : ghi (ghi đè lên tập tin cũ hoặc tạo mới nếu tập tin không có trên đĩa)
 - “ab” : ghi nối vào cuối tập tin.
 - “rb+” : đọc/ghi. Tập tin phải có trên đĩa.
 - “wb+” : tạo mới tập tin cho phép đọc ghi.
 - “ab+” : đọc, ghi vào cuối tập tin. Tạo mới tập tin nếu tập tin chưa có trên đĩa.

Đọc ghi tập tin Nhị phân

- fread() : đọc

```
size_t fread ( void *Ptr, size_t ItemSize,  
              size_t NumItem, FILE * fp );
```

fread đọc NumItem khối dữ liệu, mỗi khối có kích thước ItemSize từ fp và chứa vào vùng nhớ xác định bởi Ptr.

fread trả về số khối dữ liệu đọc được.

Nếu có lỗi hoặc EOF thì giá trị trả về nhỏ hơn NumItem

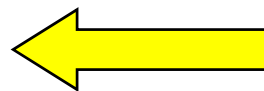
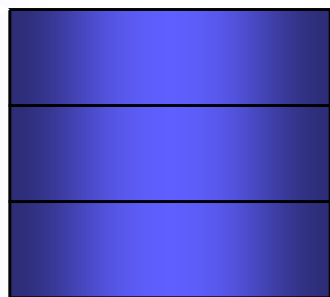
- fwrite() : ghi

```
size_t fwrite ( const void *Ptr, size_t ItemSize,  
              size_t NumItem, FILE * fp );
```

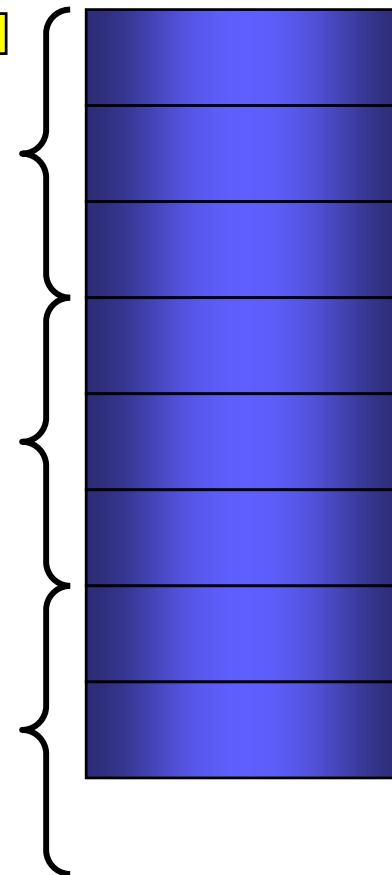
fwrite ghi khối (NumItem x ItemSize) xác định bởi Ptr ra fp.

- Chép 3 mục từ tập tin filename vào vùng nhớ trỏ bởi ptr

ptr



fp → filename



Con trỏ FILE

- Một tập tin sau khi mở được quản lý thông qua một con trỏ FILE.
- Khi mở tập tin (wb, rb), con trỏ FILE chỉ đến đầu tập tin.
- Khi mở tập tin (ab), con trỏ FILE chỉ đến cuối tập tin.
- Con trỏ FILE chỉ đến từng byte trong tập tin nhị phân.
- Sau mỗi lần đọc tập tin, con trỏ FILE sẽ di chuyển đi một số byte bằng kích thước (byte) của khối dữ liệu đọc được.

fseek(),

- Các hằng dùng trong di chuyển con trỏ FILE

```
#define SEEK_SET 0
```

```
#define SEEK_CUR 1
```

```
#define SEEK_END 2
```

- **int fseek(FILE *fp, long int offset, int whence);**

fseek di chuyển con trỏ fp đến vị trí offset theo mốc whence

- offset: khoảng cách (byte) cần di chuyển tính từ vị trí hiện tại.
(offset > 0: đi về phía cuối tập tin, offset < 0: ngược về đầu tập tin)
- whence: SEEK_SET: tính từ đầu tập tin
 SEEK_CUR: tính từ vị trí hiện hành của con trỏ
 SEEK_END: tính từ cuối tập tin
- fseek trả về: 0 nếu thành công, <>0 nếu di chuyển có lỗi

ftell() và rewind()

- rewind đặt lại vị trí con trỏ về đầu tập tin.

```
void rewind ( FILE * fp );
```

tương đương với fseek (fp, 0L, SEEK_SET);

- ftell trả về vị trí offset hiện tại của con trỏ

```
long int ftell ( FILE * fp );
```

Nếu có lỗi, ftell trả về -1L

Ví dụ: xác định kích thước tập tin.

■ ...// khai báo biến cần thận

```
1. if ( fp = fopen ( FileName, "rb" ) ) == NULL )
2.     fprintf( stderr, "Cannot open %s\n", FileName );
3. else
4.     {
5.     fseek( fp, 0, SEEK_END );
6.     FileSize = ftell( fp );
7.     printf( "File size : %d bytes\n ", FileSize );
8.     fclose( fp );
9.     }
```

Vị trí con trỏ: `fgetpos()` và `fsetpos()`

- với các tập tin có kích thước cực lớn `fseek` và `ftell` sẽ bị giới hạn bởi kích thước của `offset`.
- Dùng

```
int fgetpos ( FILE *fp, fpos_t *position);
```

```
int fsetpos ( FILE *fp, const fpos_t *position);
```

Xóa và đổi tên tập tin

- Thực hiện xóa

```
int remove(const char *filename);
```

- đổi tên tập tin

```
int rename(const char *oldname, const char *newname);
```

Chú ý khi làm việc với tập tin

- Khi mở tập tin filename, tập tin này phải nằm cùng thư mục của chương trình hoặc
- phải cung cấp đầy đủ đường dẫn đến tập tin

vd: C:\baitap\taptin.dat

viết như thế nào?

```
fp = fopen( "C:\baitap\taptin.dat", "rb" );
```

SAI RỒI

- vì có ký tự đặc biệt '\' nên viết đúng sẽ là:

```
fp = fopen( "C:\\baitap\\taptin.dat", "rb" );
```

Tham số dòng lệnh chương trình

- Khi gọi chạy một chương trình, chúng ta có thể cung cấp các tham số tại dòng lệnh gọi chương trình.

ví dụ: `dir A:*.c /w`

“copy”, “A:*.c” và “/w” là hai tham số điều khiển chương trình.

- command-line arguments.
- Các tham số dòng lệnh được tham chiếu qua hai đối số khai báo trong hàm main.

- ```
main (int argc, char * argv[])
 { ... }
```

- *argc: argument count*

- *argv: argument vector*

# Tham số dòng lệnh – ví dụ

- ... // addint.c → addint.exe
- 1. void main (int argc, char \* argv [ ])
- 2. {
- 3.     int res = 0;
- 4.     printf (“ Program %s\n “, argv[0]);
- 5.     for ( int i = 1; i < argc; i++ )
- 6.         res += atoi(argv[i]);
- 7.     printf (“ %d\n”, res );
- 8. }
- G:\>addint 3 -2 1 5 6 -2 1  
12



# Đệ quy - Recursion

Phạm Thế Bảo

Trường Đại học Khoa học Tự nhiên Tp.HCM



# Thuật toán đệ quy

- Là mở rộng cơ bản nhất của khái niệm thuật toán.
- Tư tưởng giải bài toán bằng đệ quy là đưa bài toán hiện tại về một bài toán *cùng loại, cùng tính chất* (đồng dạng) nhưng ở *cấp độ thấp hơn*, quá trình này tiếp tục cho đến khi bài toán được đưa về một cấp độ mà tại đó có thể giải được. Từ cấp độ này ta *lần ngược* để giải các bài toán ở cấp độ cao hơn cho đến khi giải xong bài toán ban đầu.
- Ví dụ:
  - định nghĩa giai thừa:  $n! = n \cdot (n-1)!$  với  $0! = 1$
  - Dãy Fibonacci:  $f_0 = 1, f_1 = 1$  và  $f_n = f_{n-1} + f_{n-2} \quad \forall n > 1$
  - Danh sách liên kết.
  - ...

- Mọi thuật toán đệ quy gồm 02 phần:

- Phần cơ sở:

- Là các trường hợp không cần thực hiện lại thuật toán (không yêu cầu gọi đệ quy). Nếu thuật toán đệ quy không có phần này thì sẽ bị lặp vô hạn và sinh lỗi khi thực hiện. Đôi lúc gọi là *trường hợp dừng*.

- Phần đệ quy:

- Là phần trong thuật toán có yêu cầu gọi đệ quy, yêu cầu thực hiện thuật toán ở một cấp độ thấp hơn.

# Các loại đệ quy

Có 03 loại đệ quy:

1. Đệ quy đuôi:

Là loại đệ quy mà trong một cấp đệ quy chỉ có duy nhất một lời gọi đệ quy xuống cấp thấp.

Ví dụ:

i. Tính giai thừa

```
giaiThua(int n){
 if(n==0)
 giaiThua = 1;
 else giaiThua= n*giaiThua(n-1);
}
```

ii. Tìm kiếm nhị phân

```
int searchBinary(int left,int right, intx){
 if(left<right){
 int mid=(left+right)/2;
 if(x==A[i])return i;
 if(x<A[i])return searchBinary(left,mid-1,x);
 return searchBinary(mid+1,right,x);
 }
 return -1;
}
```

iii. Phân tích một số nguyên ra thừa số nguyên tố (**Bài tập**)

## 2. Độ quy nhánh

Là dạng đệ quy mà trong quá trình đệ quy, lời gọi được thực hiện nhiều lần.

Ví dụ:

- i. Tháp Hà nội.
- ii. Liệt kê tất cả hoán vị của  $n$  phần tử khác nhau.

Thuật toán:

Xét tất cả các phần tử  $a_i$  với  $i=1..n$

Bỏ phần tử  $a_i$  ra khỏi dãy số

Ghi nhận đã lấy ra phần tử  $a_i$

Hoán vị (Dãy số)

Đưa phần tử  $a_i$  vào lại dãy số

Nếu (Dãy số) rỗng thì thứ tự các phần tử được lấy ra chính là một hoán vị

- iii. Bài toán tô màu (floodfill)

### 3. Đệ quy hồi tương

Là dạng đệ quy mà trong đó việc gọi có xoay vòng, như A gọi B, B gọi C, và C gọi A. Đây là trường hợp rất phức tạp.

Ví dụ:

- i. Đường Hilbert
- ii. Đường Sierpinski

# Các phương pháp khử đệ quy

1. Vòng lặp
2. Bảng stack

# Đệ quy

- Ví dụ: Viết chương trình nhập số tự nhiên  $n$  và tính giai thừa :  $n!$ .
- Giải quyết bài toán bằng vòng lặp

```
1. #include <stdio.h>
2. unsigned long int factorial(int n)
3. { unsigned long f = 1;
4. for (int i = 1; i<=n; i++)
5. f *= i;
6. return f;
7. }
8. int main(void)
9. { int n;
10. printf("Nhap n:"); scanf("%d", &n);
11. printf("n! = %d! = %l\n", n, factorial(n));
12. return 0;
}
```



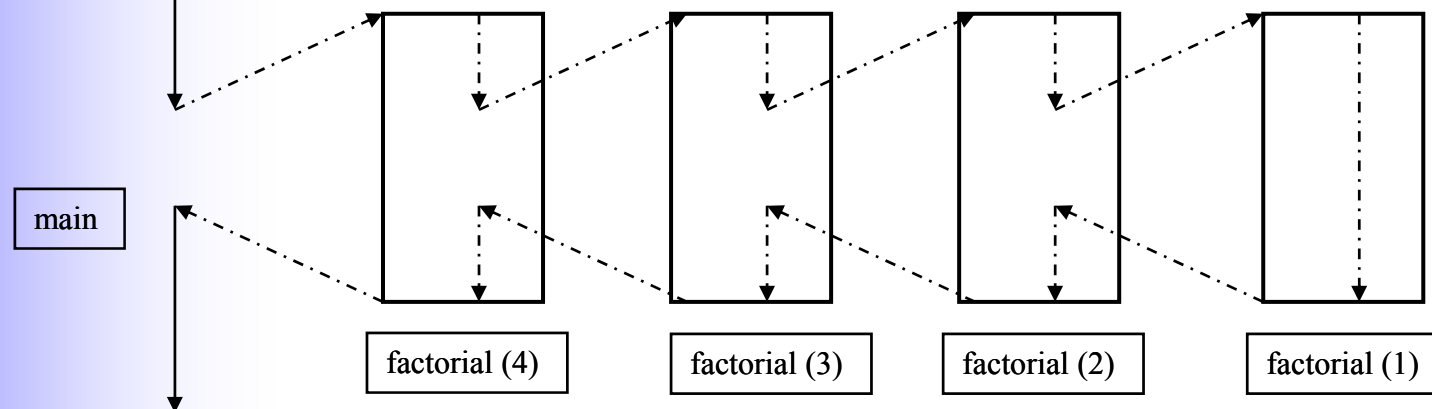
# Đệ quy

- Một hàm được gọi là đệ quy nếu như trong quá trình xử lý, hàm này có một lời gọi đến chính nó.
- Giải quyết bài toán bằng đệ quy

```
1. #include <stdio.h>
2. unsigned long int factorial(int n)
3. { if(n==0)
4. return 1;
5. return (n* factorial(n-1));
6. }
7. int main(void)
8. { int n;
9. printf("Nhap n:"); scanf("%d", &n);
10. printf("n! = %d! = %l\n", n, factorial(n));
11. return 0;
12. }
```

# Lời gọi hàm đệ quy và Điều kiện dừng của thuật giải đệ quy

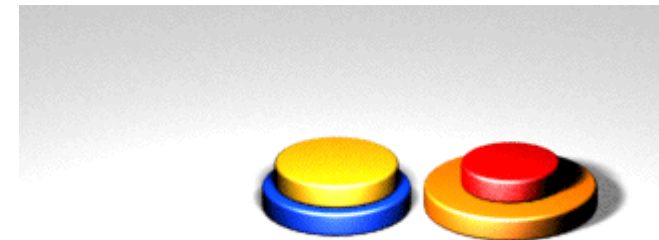
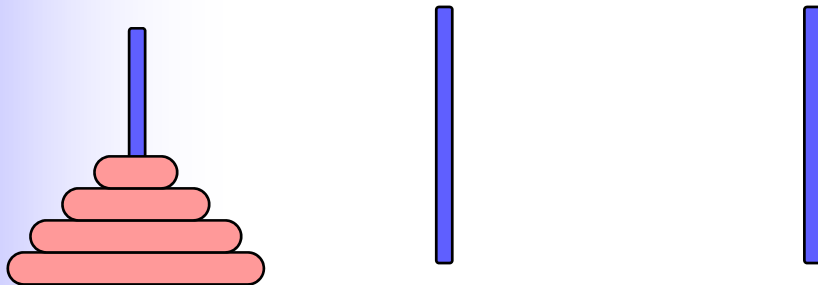
- Bài toán giải bằng thuật giải đệ quy phải có điều kiện dừng.
- Thuật toán đệ quy trên máy tính có thể bị giới hạn bởi dung lượng bộ nhớ do lời gọi hàm liên tiếp.



*Hãy vẽ sơ đồ tiến trình gọi hàm khi thực hiện tính dãy fibonacci bằng đệ quy.*

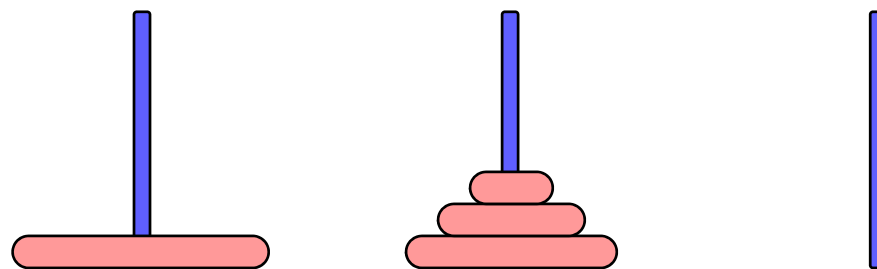
# Bài toán Tháp Hà Nội

- Có 3 cái cột và một chồng đĩa ở cột thứ nhất. Hãy chuyển chồng đĩa sang cột thứ ba với điều kiện mỗi lần di chuyển chỉ một đĩa và các đĩa bé luôn nằm trên đĩa lớn.



# Thuật giải

- Chuyển  $(n-1)$  đĩa sang cột trung gian.
- Chuyển đĩa lớn nhất sang cột đích.
- Chuyển  $(n-1)$  đĩa từ cột trung gian sang cột đích.

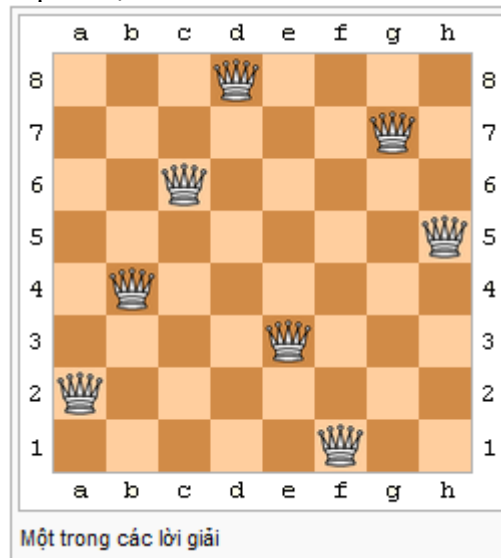


# Cài đặt bằng đệ quy

```
1. MoveDisk(disk_number, starting_post, target_post,
 intermediate_post)
2. {
3. if(disk)number > 1)
4. {
5. MoveDisk(disk_number-1, starting_post,
 intermediate_post, target_post);
6. printf("Move disk number %d, from post %d to post %d.\n",
 disk_number, starting_post, target_post);
7. MoveDisk(disk_number-1,intermediate_post,
 target_post, starting_post);
8. }
9. else
10. printf("Move disk number 1 from post %d to post %d.\n",
 starting_post, target_post);
11. }
```

## Bài toán Xếp Hậu (8 Queens).

Liệt kê tất cả các cách xếp  $n$  quân hậu trên bàn cờ  $n \times n$  sao cho chúng không ăn được nhau.



Bàn cờ có  $n$  hàng được đánh số từ  $0$  đến  $n-1$ ,  $n$  cột được đánh số từ  $0$  đến  $n-1$ ; Bàn cờ có  $n*2-1$  đường chéo xuôi được đánh số từ  $0$  đến  $2*n-2$ ,  $2*n-1$  đường chéo ngược được đánh số từ  $0$  đến  $2*n-2$ . Ví dụ: với bàn cờ  $8 \times 8$ , chúng ta có  $8$  hàng được đánh số từ  $0$  đến  $7$ ,  $8$  cột được đánh số từ  $0$  đến  $7$ ,  $15$  đường chéo xuôi,  $15$  đường chéo ngược được đánh số từ  $0$  đến  $14$ .

Vì trên mỗi hàng chỉ xếp được đúng một quân hậu, nên chúng ta chỉ cần quan tâm đến quân hậu được xếp ở cột nào. Từ đó dẫn đến việc xác định bộ  $n$  thành phần  $x_1, x_2, \dots, x_n$ , trong đó  $x_i = j$  được hiểu là quân hậu tại dòng  $i$  xếp vào cột thứ  $j$ . Giá trị của  $i$  được nhận từ  $0$  đến  $n-1$ ; giá trị của  $j$  cũng được nhận từ  $0$  đến  $n-1$ , nhưng thỏa mãn điều kiện ô  $(i, j)$  chưa bị quân hậu khác chiếu đến theo cột, đường chéo xuôi, đường chéo ngược.

Việc kiểm soát theo hàng ngang là không cần thiết vì trên mỗi hàng chỉ xếp đúng một quân hậu. Việc kiểm soát theo cột được ghi nhận nhờ dãy biến logic  $aj$  với qui ước  $aj=1$  nếu cột  $j$  còn trống,  $aj=0$  nếu cột  $j$  không còn trống. Để ghi nhận đường chéo xuôi và đường chéo ngược có chiếu tới ô  $(i, j)$  hay không, ta sử dụng phương trình  $i + j = const$  và  $i - j = const$ , đường chéo thứ nhất được ghi nhận bởi dãy biến  $bj$ , đường chéo thứ 2 được ghi nhận bởi dãy biến  $cj$  với qui ước nếu đường chéo nào còn trống thì giá trị tương ứng của nó là  $1$  ngược lại là  $0$ . Như vậy, cột  $j$  được chấp nhận khi cả 3 biến  $aj, bi+j, ci+j$  đều có giá trị  $1$ . Các biến này phải được khởi đầu giá trị  $1$  trước đó, gán lại giá trị  $0$  khi xếp xong quân hậu thứ  $i$  và trả lại giá trị  $1$  khi đưa ra kết quả.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>
#define N 8
#define D (2*N-1)
#define SG (N-1)
#define TRUE 1
#define FALSE 0
void hoanghau(int);
void inloigiai(int loigiai[]);
FILE *fp;
```

```

int A[N], B[D], C[D], loigiai[N];
int soloigiai =0;
void hoanghau(int i){
 int j;
 for (j=0; j<N; j++){
 if (A[j] && B[i-j+SG] && C[i+j]) {
 loigiai[i]=j;
 A[j]=FALSE;
 B[i-j+SG]=FALSE;
 C[i+j]=FALSE;
 if (i==N-1){
 soloigiai++;
 inloigiai(loigiai);
 delay(500);
 }
 else
 hoanghau(i+1);
 A[j]=TRUE;
 B[i-j+SG]=TRUE;
 C[i+j]=TRUE;
 }
 }
}

void inloigiai(int *loigiai){
 int j;
 printf("\n Lờ i giải %3d:", soloigiai);
 fprintf(fp, "\n Lờ i giải %3d:", soloigiai);
 for (j=0; j<N; j++){
 printf("%3d", loigiai[j]);
 fprintf(fp, "%3d", loigiai[j]);
 }
}

void main(void){
 int i;
 clrscr();
 fp=fopen("loigiai.txt", "w");
 for (i=0; i<N; i++){
 A[i]=TRUE;
 }
 for(i=0; i<D; i++){
 B[i]=TRUE;
 C[i]=TRUE;
 }
 hoanghau(0);
 fclose(fp);
}

```

---

## Bài toán Tháp Hà Nội (Tower of Hanoi)

Truyền thuyết 1: Một nhà toán học Pháp sang thăm Đông Dương đến một ngôi chùa cổ ở Hà Nội thấy các vị sư đang chuyển một chồng đĩa quý gồm 64 đĩa với kích thước khác nhau từ cột A sang cột C theo cách:

- Mỗi lần chỉ chuyển 1 đĩa .
- Khi chuyển có thể dùng cột trung gian B .
- Trong suốt quá trình chuyển các chồng đĩa ở các cột luôn được xếp đúng (đĩa có kích thước bé được đặt trên đĩa có kích thước lớn) .

Khi được hỏi các vị sư cho biết khi chuyển xong chồng đĩa thì đến ngày tận thế !

Truyền thuyết 2: Lúc thế giới hình thành, trong ngôi đền thờ Brahma có một chồng 64 cái đĩa. Mỗi ngày, có một thầy tu di chuyển một đĩa. Đến khi hết đĩa thì đó là ngày tận thế.

Như sẽ chỉ ra sau này với chồng gồm  $n$  đĩa cần  $2^n - 1$  lần chuyển cơ bản (chuyển 1 đĩa).  $2n$  Giả sử thời gian để chuyển 1 đĩa là  $t$  giây thì thời gian để chuyển xong chồng 64 đĩa sẽ là:  
 $T = (2^{64} - 1) * t \text{ giây} = 11.84 * 10^{19} * t \text{ giây}$

Với  $t = 1/100$  giây thì  $T = 5.8 * 10^9$  năm = 5.8 tỷ năm.

Ta có thể tìm thấy giải thuật (dãy các thao tác cơ bản) cho bài toán một cách dễ dàng ứng với trường hợp chồng đĩa gồm 0, 1, 2, 3 đĩa. Với chồng 4 đĩa giải thuật bài toán đã trở nên phức tạp. Tuy nhiên giải thuật của bài toán lại được tìm thấy rất dễ dàng nhanh chóng khi ta khái quát số đĩa là  $n$  bất kỳ và nhìn bài toán bằng quan niệm đệ quy.

### a) Thông số hóa bài toán.

Xét bài toán ở mức tổng quát nhất: chuyển  $n$  ( $n \geq 0$ ) đĩa từ cột X sang cột Z lấy cột Y làm trung gian.

Ta gọi giải thuật giải bài toán ở mức tổng quát là thủ tục THN( $n, X, Y, Z$ ) chứa 4 thông số  $n, X, Y, Z$ ;  $n$  thuộc tập số tự nhiên  $N$  (kiểu nguyên không dấu);  $X, Y, Z$  thuộc tập các ký tự (kiểu ký tự).

Bài toán cổ ở trên sẽ được thực hiện bằng lời gọi THN(64, A, B, C).

Dễ thấy rằng: trong 4 thông số của bài toán thì thông số  $n$  là thông số quyết định độ phức tạp của bài toán ( $n$  càng lớn thì số thao tác chuyển đĩa càng nhiều và thứ tự thực hiện chúng càng khó hình dung),  $n$  là thông số điều khiển.

### b) Trường hợp suy biến và cách giải.

Với  $n = 1$  bài toán tổng quát suy biến thành bài toán đơn giản THN(1, X, Y, Z): tìm dãy thao tác để chuyển chồng 1 đĩa từ cột X sang cột Z lấy cột Y làm trung gian. Giải thuật giải bài toán THN(1, X, Y, Z) là thực hiện chỉ 1 thao tác cơ bản: Chuyển 1 đĩa từ X sang Z (ký hiệu là Move(X, Z)).  $\text{THN}(1, X, Y, Z) \equiv \{ \text{Move}(X, Z) \}$

Chú ý: Hoàn toàn tương tự ta cũng có thể quan niệm trường hợp suy biến là trường hợp  $n = 0$  tương ứng với bài toán THN(0, X, Y, Z): chuyển 0 đĩa từ X sang Z lấy Y làm trung gian mà giải thuật tương ứng là không làm gì cả (thực hiện thao tác rỗng).

$\text{THN}(0, X, Y, Z) \equiv \{ \phi \}$

### c) Phân rã bài toán:

Ta có thể phân rã bài toán THN( $k, X, Y, Z$ ): chuyển  $k$  đĩa từ cột X sang cột Z lấy cột Y làm trung gian thành dãy tuần tự 3 công việc sau:



+ Chuyển (k - 1) đĩa từ cột X sang cột Y lấy cột Z làm trung gian :  
 THN (k - 1, X, Z, Y) (bài toán THN với n = k-1, X = X , Y = Z , Z = Y )  
 + Chuyển 1 đĩa từ cột X sang cột Z : Move ( X , Z ) (thao tác cơ bản ).  
 + Chuyển (k - 1) đĩa từ cột Y sang cột Z lấy cột X làm trung gian :  
 THN (k - 1, Y, X, Z) ( bài toán THN với n = k-1 , X = Y , Y = X , Z = Z ) .  
 Vậy giải thuật trong trường hợp tổng quát (n > 1) là :

```
THN(n,X,Y,Z) ≡ {
 THN (n -1,X,Z,Y) ;
 Move (X , Z) ;
 THN (n -1,Y,X,Z) ;
}
```

Với n đĩa thì cần bao nhiêu bước chuyển 1 đĩa? Thực chất trong thủ tục THN các lệnh gọi đệ qui chỉ nhằm sắp xếp trình tự các thao tác chuyển 1 đĩa

Số lần chuyển 1 đĩa được thực hiện là đặc trưng cho độ phức tạp của giải thuật .

Với n đĩa , gọi f(n) là số các thao tác chuyển \_một\_ đĩa .

Ta có :        f(0) = 0 .  
                f(1) = 1 .  
                f(n) = 2f(n - 1) + 1 với n > 0

Do đó :        f(n) = 1 + 2 + 2<sup>2</sup> + + 2<sup>n-1</sup> = 2<sup>n</sup> - 1

Để chuyển 64 đĩa cần 2<sup>64</sup> - 1 bước hay xấp xỉ 10<sup>20</sup> bước . Cần khoảng 10 triệu năm với một máy tính nhanh nhất hiện nay để làm việc này .

### **Hàm thực hiện:**

```
void THN(int n, char X,Y,Z){
 if(n > 0) {
 THN(n -1,X,Z,Y) ;
 Move (X , Z) ;
 THN(n - 1,Y,X,Z) ;
 }
 return ;
}
```

}

hoặc :

```
void THN(int n , char X,Y,Z) {
 if(n == 1) Move (X , Z) ;
 else {
 THN(n -1,X,Z,Y) ;
 Move (X , Z) ;
 THN(n - 1,Y,X,Z) ;
 }
 return ;
}
```

}

---

**Bài toán tìm cặp điểm gần nhất trong mặt phẳng (*Closest Pair*).**

# Bài tập

## Phần 1: Mảng và chuỗi ký tự

1. Nhập vào một dãy số nguyên gồm  $n$  phần tử. Tìm cặp phần tử có tổng đúng bằng  $k$  ( $k$  nhập từ bàn phím).
2. Nhập dãy  $n$  số ( $n \leq 1000$ ). Xác định đường chạy dài nhất, xuất lên màn hình vị trí phần tử đầu tiên và độ dài của đường chạy đó. Đường chạy là một dãy liên tiếp các phần tử không giảm của dãy ban đầu.  
Ví dụ : Nhập dãy 1 4 2 3 1 2 6 8 3 5 7  
Đường chạy dài nhất là : 4 4
3. Nhập dãy  $n$  số ( $n \leq 1000$ ). Xét dãy số có đối xứng không?
4. Viết chương trình nhập/ xuất một ma trận số thực  $n \times m$  ( $n, m \leq 100$ ). Sắp xếp các giá trị các phần tử của ma trận không giảm theo đường zigzag  
Ví dụ:  

|       |         |       |
|-------|---------|-------|
| 1 2 3 |         | 1 2 3 |
| 4 5 6 | kết quả | 6 5 4 |
| 7 8 9 |         | 7 8 9 |
5. Viết chương trình nhập/ xuất một ma trận vuông số nguyên dương  $n \times n$  ( $n \leq 100$  và  $n$  lẻ). Xét ma trận có đối xứng qua:
  - a. Đường chéo chính.
  - b. Theo dòng, cột.
  - c. Tâm.
6. Viết chương trình nhập chuỗi ký S:
  - a. Đếm và cho biết số lượng khoảng trắng, số lượng ký số, số lượng chữ cái latin, số lượng các ký tự khác.
  - b. Đếm và cho biết số lượng từ của chuỗi – các từ cách nhau bởi khoảng trắng.
  - c. Biến đổi chuỗi sao cho các ký tự đầu mỗi từ là ký tự in hoa, các ký tự khác in thường.
7. Viết chương trình nhập chuỗi ký S, đếm và in cho biết số lượng của mỗi chữ cái latin trong chuỗi (không phân biệt chữ in hoa và chữ in thường).
8. Một số tự nhiên là Palindrom nếu các chữ số của nó viết theo thứ tự ngược lại thì số tạo thành là chính số đó ( Ví dụ: 4884, 393). Hãy tìm: Tất cả các số tự nhiên nhỏ hơn 100 mà khi bình phương lên thì cho một Palindrom.
9. Viết chương trình đảo ngược vị trí các từ trong câu. Ví dụ: “KIEN AN CA” đổi thành “CA AN KIEN”.
10. Nhập một câu không quá 20 từ, mỗi từ không quá 10 ký tự. Viết chương trình tách các từ trong câu và in các từ theo thứ tự Alphabet. Ví dụ: “PHAN VIEN CONG NGHE THONG TIN” tách thành [PHAN], [VIEN], [CONG], [NGHE], [THONG], [TIN] và in ra: CONG NGHE PHAN THONG TIN VIEN

## Phần 2: Cấu trúc

Xây dựng cấu trúc điểm, đường thẳng, hình chữ nhật, đường tròn.

1. Xét vị trí tương đối của 03 điểm trên mặt phẳng.
2. Xét vị trí tương đối giữa hai đoạn thẳng.
3. Xét 1 điểm có nằm trong 1 : hình chữ nhật, tròn hay không
4. Xét vị trí tương đối của 1 đường thẳng và hình chữ nhật, tròn

5. Viết chương trình nhập thông tin của một sinh viên, xuất thông tin sinh viên vừa nhập ra màn hình. Thông tin một sinh viên gồm: Mã sinh viên (chuỗi 8 ký tự), họ và tên sinh viên (chuỗi 30 ký tự), giới tính (nam/nữ), địa chỉ liên hệ (chuỗi 50 ký tự), điểm 6 môn học.
6. Viết chương trình quản lý một lớp học gồm tối đa 150 sinh viên, mỗi sinh viên có các thông tin như bài trước. Chương trình phải đảm bảo một số tính năng:
  - a. Nhập mới một danh sách sinh viên.
  - b. Tìm một sinh viên trong danh sách theo mã sinh viên.
  - c. Thêm một sinh viên vào danh sách.
  - d. Hủy một sinh viên ra khỏi danh sách.
  - e. Xuất danh sách sinh viên ra màn hình.
  - f. Xuất danh sách các sinh viên còn nợ điểm (điểm < 5) của ít nhất một môn học.

### Phần 3: Con trỏ và tập tin

1. Viết chương trình ghi vào tập tin sochan.dat các số nguyên chẵn từ 0 đến 100.
2. Cho mảng các số nguyên, tính tổng các phần tử của mảng.  
Dữ liệu vào : tập tin văn bản ARRAY.INP gồm hai dòng
  - Dòng 1 chứa số nguyên  $n$  mô tả số phần tử của mảng
  - Dòng 2 chứa  $n$  số nguyên
 Kết quả : Đưa ra tập tin văn bản ARRAY.OUT gồm một dòng ghi tổng các phần tử trong mảng.
3. Viết chương trình sao chép 2 file văn bản
4. Viết chương trình giả lập lệnh copy con của dos để tạo file văn bản.
5. Cho mảng các số nguyên, hãy liệt kê các phần tử là số nguyên tố  
Dữ liệu vào : tập tin văn bản NT.INP gồm hai dòng
  - Dòng 1 chứa số nguyên  $n$  ( $n \leq 100$ )
  - Dòng 2 chứa  $n$  số nguyên
 Kết quả : đưa ra tập tin văn bản NT.OUT gồm hai dòng:
  - Dòng 1 chứa số lượng các phần tử nguyên tố trong mảng.
  - Dòng 2 liệt kê các số nguyên tố đó.
6. Tạo file văn bản có tên là "INPUT.TXT" có cấu trúc như sau:
  - Dòng đầu tiên ghi  $N$  ( $N$  là số nguyên dương nhập từ bàn phím).
  - Trong các dòng tiếp theo ghi  $N$  số nguyên ngẫu nhiên trong phạm vi từ 0 đến 100, mỗi dòng 10 số (các số cách nhau ít nhất một khoảng trắng). Hãy đọc dữ liệu của file "INPUT.TXT" và lưu vào mảng một chiều A.
 Thực hiện các công việc sau :
  - Tìm giá trị lớn nhất của mảng A.
  - Đếm số lượng số chẵn, số lượng số lẻ của mảng A.
  - Hãy sắp xếp các phần tử theo thứ tự tăng dần.
  - Hãy ghi các kết quả vào file văn bản có tên OUTPUT.TXT theo mẫu sau

| INPUT.TXT |    |    |    |    |    |    |    |    |  |
|-----------|----|----|----|----|----|----|----|----|--|
| 18        |    |    |    |    |    |    |    |    |  |
| 87        | 39 | 78 | 19 | 89 | 4  | 40 | 98 | 29 |  |
| 65        |    |    |    |    |    |    |    |    |  |
| 20        | 43 | 1  | 99 | 38 | 34 | 58 | 4  |    |  |

| OUTPUT.TXT |    |    |    |    |    |    |    |    |    |
|------------|----|----|----|----|----|----|----|----|----|
| Cau a: 99  |    |    |    |    |    |    |    |    |    |
| Cau b: 9 9 |    |    |    |    |    |    |    |    |    |
| Cau c:     |    |    |    |    |    |    |    |    |    |
| 1          | 4  | 4  | 19 | 20 | 29 | 34 | 38 | 39 | 40 |
| 43         | 58 | 65 | 78 | 87 | 89 | 98 | 99 |    |    |

7. Hoàn thiện phần 2, câu 6 bằng cách lưu trữ dữ liệu mảng sinh viên vào tập tin nhị phân có tên `sinhvien.dat`. Và khi đọc dữ liệu từ tập tin `sinhvien.dat` sẽ được đưa vào một mảng một chiều (dùng con trỏ để khai báo) sẽ được cấp phát số lượng phần tử linh động tùy thuộc vào số lượng phần tử trong tập tin.

### **Phần 4: Đệ quy**

1. Tính tổng giá trị các phần tử của một mảng bằng phương pháp đệ quy.
2. Cài đặt bài toán tám hậu và vẽ bàn cờ tương ứng với các vị trí đặt hậu.
3. Cài đặt bài toán tháp Hà Nội với số đĩa nhỏ hơn 5.