CHƯƠNG 3
LẬP TRÌNH HÀM VÀ ĐỆ QUY

GIẢNG VIÊN

TS. Hà Ngọc Long

NỘI DUNG MÔN HỌC

1. Giới thiệu về môn học
2. Cấu trúc điều khiển
3. Lập trình hàm và đệ quy
4. Mảng
5. Xây ký tự
6. Con trỏ
7. Kiểu cấu trúc
8. Con trỏ và danh sách liên kết đơn

TÀI LIỆU HỌC TẬP

- Trương Công Tuấn. Ngôn ngữ lập trình C – Các vấn đề cốt yếu, NXB Đại học Huế, 2008.
- Nguyễn Thanh Thủy. Ngôn ngữ C. NXB Khoa học và Kỹ thuật, 1999.
- Phạm Văn Ất. Kỹ thuật lập trình C. NXB Khoa học và Kỹ thuật, 1995.

Tài liệu hướng dẫn và phần mềm thực hành:

- Dev-C++ (IDE)
- Visual Studio Code (IDE)
- Videos bài giảng và hướng dẫn thực hành

Hàm và Tham Số

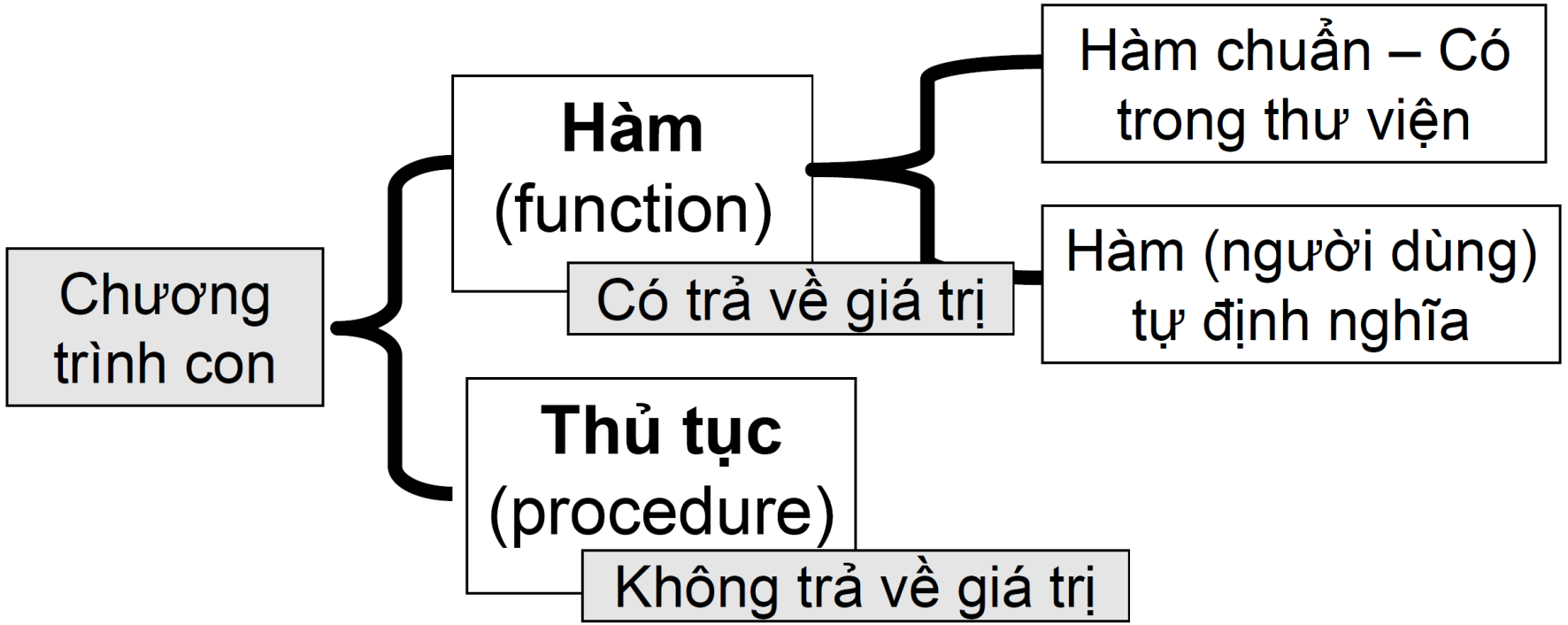
Nội dung chính

1. **Khái niệm hàm**
   - Khái niệm chương trình con
   - Phân loại: hàm và thủ tục
2. **Khai báo và sử dụng hàm**
   - Khai báo và sử dụng
3. **Phạm vi của biến**
   - Toàn cục và địa phương
4. **Truyền tham số**
   - Truyền theo gia trị, truyền theo địa chỉ
   - Biến **`static`**, biến **`register`**

Khái niệm và Vai trò

- **Khái niệm (hàm)**
  - Là một chương trình nằm trong một chương trình lớn hơn nhằm thực hiện một nhiệm vụ cụ thể
- **Vai trò**
  - **Chia nhỏ** chương trình ra thành từng phần để quản lý
    - Phương pháp lập trình có cấu trúc
  - Có thể **sử dụng lại** nhiều lần: `printf(), scanf()`…
  - Chương trình **dễ dàng đọc và bảo trì** hơn

Phân loại



- **Ngôn ngữ lập trình C**
  - Chỉ cho phép khai báo chương trình con là hàm.
  - Sử dụng kiểu "**`void`**" với ý nghĩa "không là kiểu dư liệu nào cả" để chuyển thủ tục về dạng hàm

Tại sao cần sử dụng hàm?

- Chia vấn đề thành nhiều tác vụ con
  - Dễ dàng hơn khi giải quyết các vấn đề phức tạp
- Tổng quát hóa được tập các câu lệnh hay lặp lại
  - Ta không phải viết cùng một thứ lặp đi lặp lại nhiều lần
  - **`printf`** và **`scanf`** là ví dụ điển hình…
- Hàm giúp chương trình dễ đọc và bảo trì hơn nhiều

Nội dung chính

1. **Khái niệm hàm**
   - Khái niệm chương trình con
   - Phân loại: hàm và thủ tục
2. **Khai báo và sử dụng hàm**
   - Khai báo và sử dụng
3. **Phạm vi của biến**
   - Toàn cục và địa phương
4. **Truyền tham số**
   - Truyền theo gia trị, truyền theo địa chỉ
   - Biến **`static`**, biến **`register`**

Ví dụ

- **Khái niệm**
  - Là một chương trình nằm trong một chương trình lớn hơn nhằm thực hiện một nhiệm vụ cụ thể
- **Vai trò**
  - Chia nhỏ chương trình ra thành từng phần để quản lý.
    - Phương pháp lập trình có cấu trúc.
  - Có thể sử dụng lại nhiều lần: `printf()`, `scanf()`...
  - Chương trình dễ dàng đọc và bảo trì hơn.

Ví dụ

Khai báo chương trình con

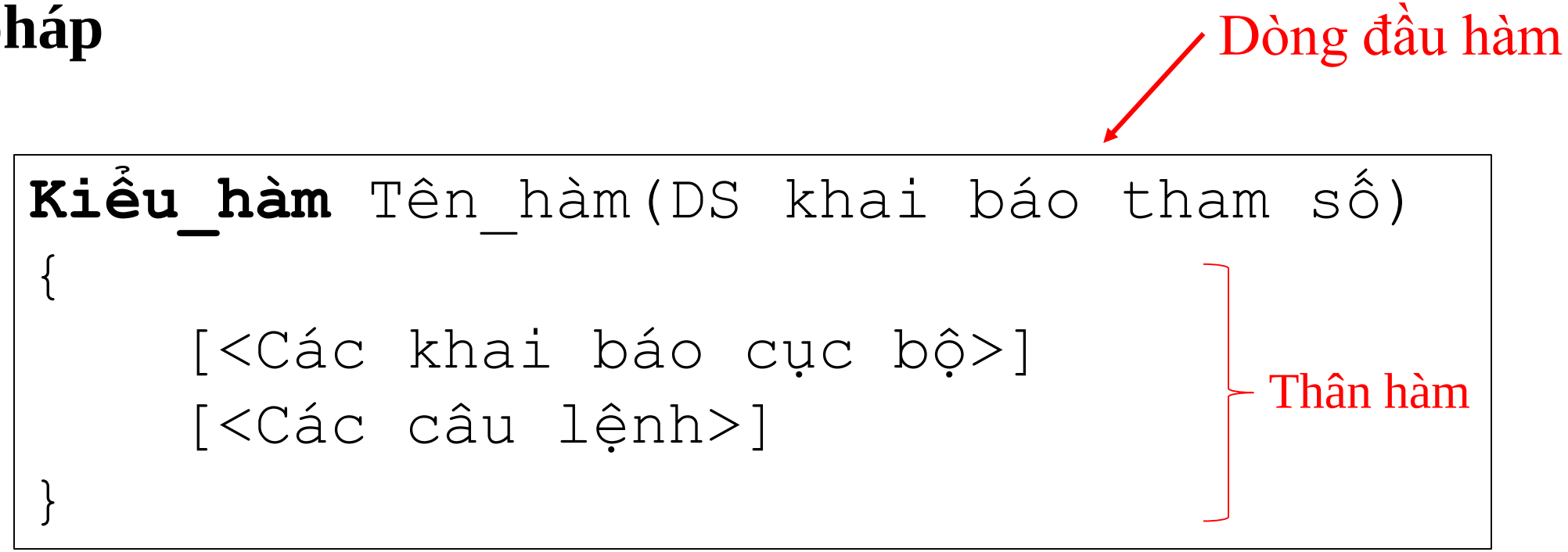
Gọi chương trình con ra thực hiện

```
#include<stdio.h>
int bp(int x){
    int y;
    y = x * x;
    return y;
}
void main(){
    int i;
    for (i=1; i< 20; i+=2)
        printf("%4d\n", bp(i));
    printf("\n");
}
```

```
   1
   9
  25
  49
  81
 121
 169
 225
 289
 361
```

Định nghĩa hàm

- **Cú pháp**



Dòng đầu hàm

```
Kiểu_hàm Tên_hàm(DS khai báo tham số)
```

- Mô tả các thông tin được trao đổi giữa bên trong và bên ngoài hàm.
  - Tên của hàm,
  - Các tham số đầu vào
    - Hàm cần những thông tin gì để hoạt động
  - Tham số đầu ra và giá trị trả về
    - Hàm cung cấp những thông tin gì cho môi trường
- Dùng phân biệt các hàm với nhau,
  - không tồn tại 2 hàm có dòng đầu hàm giống nhau.

Tên hàm

- Là tên do người sử dụng tự định nghĩa
  - Tuân theo quy tắc đặt tên đối tượng
  - Nên mang ý nghĩa gợi ý chức năng của hàm

Khai báo tham số hình thức (1/2)

- Khai báo các thông tin cần cho hoạt động của hàm và các thông tin, kết quả tính toán được hàm trả lại.
  - Tham số chứa dữ liệu vào cung cấp cho hàm
  - Tham số chứa dữ liệu ra mà hàm tính toán được.
- Các tham số sử dụng trong khai báo hàm là tham số hình thức.
  - Nguyên tắc khai báo tham số hình thức như giống như khai báo một biến

```
kiểu_dữ_liệu_của_tham_số tên_của_tham_số
```

Khai báo tham số hình thức (2/2)

- Các tham số cung cấp cho hàm trong quá trình thực hiện hàm là tham số thực sự
  - Kiểu dữ liệu của tham số thực phải giống kiểu dữ liệu của tham số hình thức tương ứng với tham số thực sự đó,.
- Một hàm có thể có một, nhiều hoặc không có tham số nào cả
  - Nếu có nhiều tham số, phải được phân cách với nhau bằng dấu phẩy.
  - Nếu không có tham số vẫn phải có cặp dấu ngoặc đơn sau tên hàm

Kiểu dữ liệu trả về (1/2)

- Thông thường hàm sau khi được thực hiện sẽ trả về một giá trị kết quả tính toán nào đó.
- Để sử dụng được giá trị đó cần phải biết nó thuộc kiểu dữ liệu gì.
  - Kiểu dữ liệu của đối tượng tính toán được hàm trả về được gọi là kiểu dữ liệu trả về của hàm.

Kiểu dữ liệu trả về (2/2)

- Trong C, kiểu dữ liệu trả về của hàm có thể là kiểu dữ liệu bất kì (kiểu dữ liệu có sẵn hoặc kiểu dữ liệu do người dùng tự định nghĩa) nhưng không được là kiểu dữ liệu mảng.
- Nếu kiểu dữ liệu trả về là kiểu void thì hàm không trả về giá trị nào cả.
- Nếu không khai báo kiểu dữ liệu trả về thì chương trình dịch của C sẽ ngầm hiểu rằng kiểu dữ liệu trả về của hàm là kiểu `int`.

Thân hàm (1/2)

- Danh sách các câu lệnh
- Thường có ít nhất một lệnh return
- **Hoạt động của hàm**
- Thực hiện lần lượt các lệnh cho đến khi
  - Thực hiện xong tất cả các câu lệnh có trong thân hàm
  - Gặp lệnh return
    - Cú pháp chung

```
return [biểu_thức];
```

Thân hàm (2/2)

- Khi gặp lệnh `return biểu_thức`
- Tính toán giá trị của `biểu_thức`,
- Lấy kết quả tính toán được làm giá trị trả về cho lời gọi hàm
- Kết thúc việc thực hiện hàm, trở về chương trình đã gọi nó.
- Nếu `return` không có phần `biểu_thức`,
  - Kết thúc thực hiện hàm mà không trả về giá trị nào cả.
    - Dùng khi hàm được khai báo có kiểu trả về là `void`

Sử dụng hàm

```
Tên_hàm (DS_tham_số_thực _sự);
```

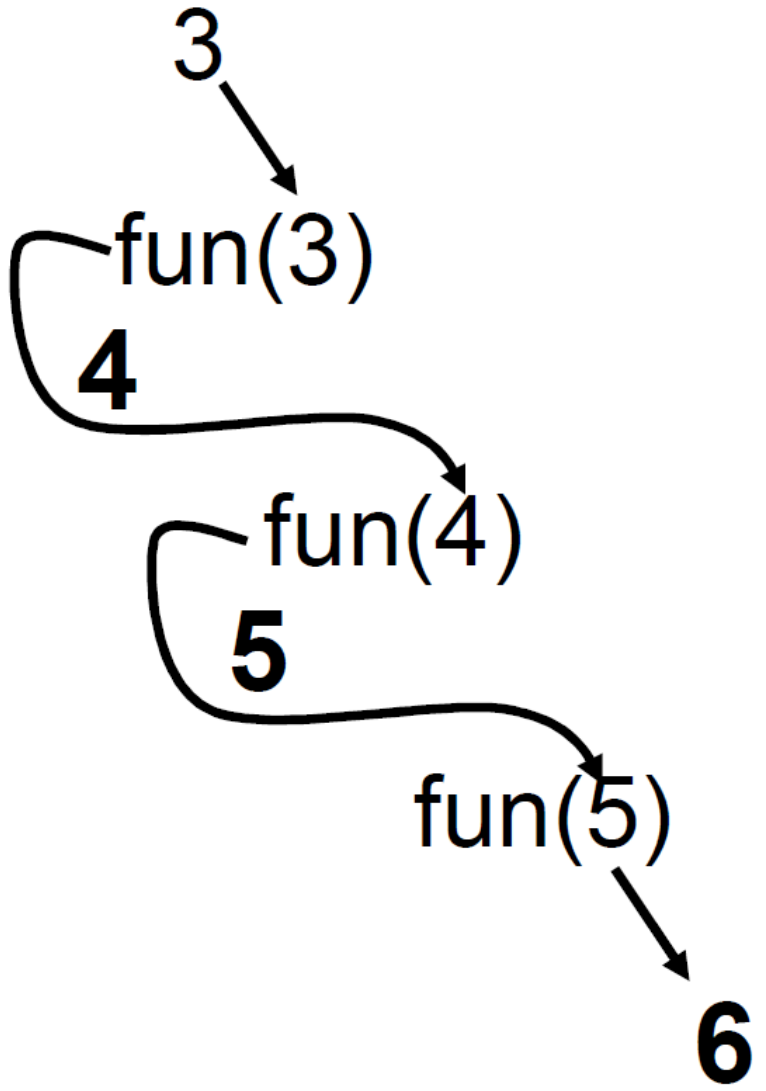
- **Ví dụ**:

```
N = bp(1);N= bp(3);,…
```

- **Lưu ý:**
  - Gọi hàm thông qua tên hàm và các tham số được cung cấp thực sự cho hàm (*tham số thực sự*).
  - Nếu hàm nhận nhiều tham số thì các tham số ngăn cách nhau bởi dấu phẩy
  - Các tham số hình thức của hàm sẽ nhận các giá trị từ tham số truyền vào
  - Sau khi thực hiện xong, trở về điểm mà hàm được gọi

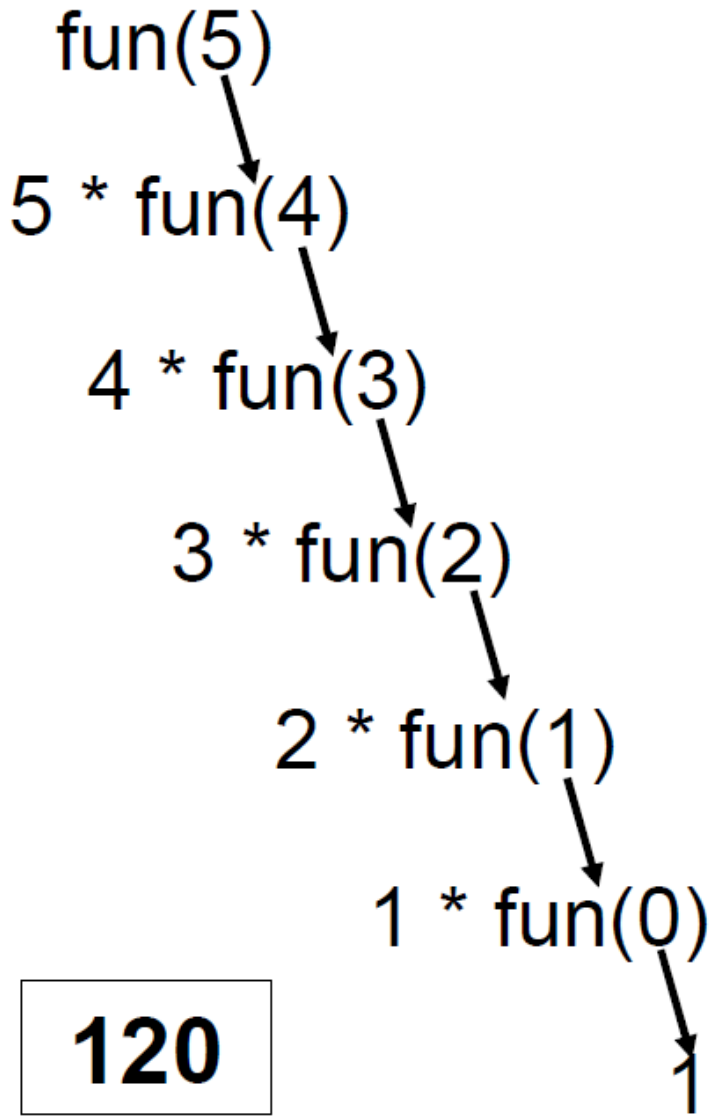
Ví dụ: Hãy cho biết kết quả của chương trình

```c
#include<stdio.h>
int fun(int a){
    a++;
    return a;
}
int main(){
    printf("%d\n", fun(fun(fun(3))));
    return 0;
}
```



Ví dụ: Hãy cho biết kết quả của chương trình

```c
#include<stdio.h>
int fun(int n){
    if(n==0) return 1;
    else return n*fun(n-1);
}
int main(){
    printf("%d\n", fun(5));
    return 0;
}
```



Ví dụ: Tính TBC f(a), f(b), f(c) nếu f(x) = $x^5 + \sqrt[5]{x}$

```
So 3 so thuc : 10 11 12
Ket qua 169962.609375
```

```c
#include <stdio.h>
#include <math.h>
float f(float x){
    if(x==0.0)
        return 0;
    else
        return pow(x,5)+x/fabs(x) * pow(fabs(x), 0.2);
}
void main(){
    float a, b, c;
    printf("So 3 so thuc : "); scanf("%f%f%f",&a,&b,&c);
    printf("Ket qua %f \n",(f(a)+f(b)+f(c))/3);
}
```

Ví dụ: Tính diện tích tam giác ABC

```c
#include <stdio.h>
#include <math.h>
typedef struct{
    float x, y;
}Point;
float kc(Point A, Point B){
    return sqrt(pow(A.x-B.x,2)+pow(A.y-B.y,2));}
void main(){
    Point A, B, C;
    float AB,BC,CA,p,S;
    printf("Toa do A (x,y) :"); scanf("%f%f",&A.x,&A.y);
    printf("Toa do B (x,y) :"); scanf("%f%f",&B.x,&B.y);
    printf("Toa do C (x,y) :"); scanf("%f%f",&C.x,&C.y);
    AB = kc(A,B); BC = kc(B,C); CA = kc(C,A);
    p= (AB + BC + CA)/2;
    S = sqrt(p*(p-AB)*(p-BC)*(p-CA));
    printf("Dien tich ABC %f",S);
}
```

```
Toa do A (x,y) :10 20
Toa do B (x,y) :5 5
Toa do C (x,y) :-5 -10
Dien tich ABC 37.499588
```

Ví dụ: Tìm ước chung lớn nhất của dãy số

```c
# include <stdio.h>
int uscln(int a, int b) {
    while (a !=b){
        if(a > b) a = a- b;
        else b = b - a;
    }
    return a;}
void main(){
    int A[100], N, i, r;
    printf("So phan tu : "); scanf("%d",&N);
    for(i=0; i < N; i++){
        printf("A[%d] = ",i+1); scanf("%d",&A[i]);
    }
    r = A[0];
    for(i = 1; i < N; i++)
        r = uscln(r,A[i]);
    printf("Ket qua %d \n",r);
}
```

```
So phan tu : 3
A[1] = 9
A[2] = 21
A[3] = 27
Ket qua 3
```

Nội dung chính

1. **Khái niệm hàm**
   - Khái niệm chương trình con
   - Phân loại: hàm và thủ tục
2. **Khai báo và sử dụng hàm**
   - Khai báo và sử dụng
3. **Phạm vi của biến**
   - Toàn cục và địa phương
4. **Truyền tham số**
   - Truyền theo gia trị, truyền theo địa chỉ
   - Biến **`static`**, biến **`register`**

Phạm vi

- **Phạm vi:**
  - Khối lệnh, chương trình con, chương trình chính
- Biến chỉ có tác dụng trong phạm vi được khai báo
- Trong cùng một phạm vi các biến phải có tên khác nhau.
- Tình huống
  - Trong hai phạm vi khác nhau có hai biến cùng tên. Trong đó một phạm vi này nằm trong phạm vi kia?

```
#include<stdio.h>
#include<conio.h>
int i;
int binhphuong(int x){
    int y;
    y = x * x;
    return y;
}
void main(){
    int y;
    for (i=0; i<= 10; i++){
        y = binhphuong(i);
        printf("%d ", y);
    }
}
```

Phân loại biến

- **Biến toàn cục:**
  - Biến được khai báo trong chương trình chính, được đặt sau khai báo tệp tiêu đề.
- **Biến cục bộ:**
  - Biến được khai báo trong lệnh khối hoặc chương trình con, được đặt trước các câu lệnh.
- **Ghi chú**
  - Hàm `main()` cũng là một chương trình con nhưng là nơi chương trình được bắt đầu. Biến khai báo trong hàm `main()` cũng là biếncục bộ, chỉ có phạm vi trong hàm `main()`.

Biến `static`

- Biến cục bộ ra khỏi phạm vi thì bộ nhớ dành cho biến được giải phóng
- Yêu cầu lưu trữ giá trị của biến cục bộ một cách lâu dài => sử dụng từ khóa `static`
- ***Cú pháp:***

```
static <kiểu_dữ_liệu> tên_biến;
```

Ví dụ

```c
#include <stdio.h>
# include <conio.h>
void fct() {
    static int count = 1;
    printf("\n Day la lan goi ham fct lan thu %2d", count++);
}
void main(){
    int i;
    for(i = 0; i < 10; i++)
        fct();
    getch();
}
```

```
Day la lan goi ham fct lan thu  1
Day la lan goi ham fct lan thu  2
Day la lan goi ham fct lan thu  3
Day la lan goi ham fct lan thu  4
Day la lan goi ham fct lan thu  5
Day la lan goi ham fct lan thu  6
Day la lan goi ham fct lan thu  7
Day la lan goi ham fct lan thu  8
Day la lan goi ham fct lan thu  9
Day la lan goi ham fct lan thu 10
```

Biến `register`

- Thanh ghi có tốc độ truy cập nhanh hơn RAM, bộ nhớ ngoài
- Lưu biến trong thanh ghi sẽ tăng tốc độ thực hiện chương trình
- **Cú pháp**

```
register <kiểu_dữ_liệu> tên_biến;
```

- **Lưu ý:**
  - số lượng biến `register` không nhiều và thường chỉ với kiểu dữ liệu nhỏ như `int`, `char`

Nội dung chính

1. **Khái niệm hàm**
   - Khái niệm chương trình con
   - Phân loại: hàm và thủ tục
2. **Khai báo và sử dụng hàm**
   - Khai báo và sử dụng
3. **Phạm vi của biến**
   - Toàn cục và địa phương
4. **Truyền tham số**
   - Truyền theo gia trị, truyền theo địa chỉ
   - Biến **`static`**, biến **`register`**

Ví dụ

```c
#include<stdio.h>
void swap(int a, int b) {
    int x = a;
    a = b;
    b = x;
    return;
}
void main(){
    int a = 5, b = 100;
    printf("Truoc: a=%d, b=%d \n\n",a,b);
    swap(a,b);
    printf("Sau : a=%d, b=%d\n\n",a,b);
    return;
}
```

```
Truoc: a=5, b=100

Sau : a=5, b=100
```

Hàm và truyền tham số

- ***Trong C:*** tên hàm phải là duy nhất, lời gọi hàm phải có các đối số đúng bằng và hợp tương ứng về kiểu với tham số trong đn hàm.
  - C chỉ có duy nhất 1 cách truyền tham số: tham trị (kể cả dùng địa chỉ cũng vậy).
- ***Trong C++:*** ngoài truyền tham trị, nó còn cho phép truyền tham chiếu.
  - Tham số trong C++ còn có kiểu tham số ngầm định (default parameter), vì vậy số đối số trong lời gọi hàm có thể ít hơn tham số định nghĩa.
  - Đồng thời C++ còn có cơ chế đa năng hóa hàm, vì vậy tên hàm không phải duy nhất.

Truyền theo giá trị và truyền theo biến

- **Truyền theo trị (giá trị)**
  - Dựa trên nguyên tắc ***truyền những bản sao*** của biến được truyền.
  - Những câu lệnh ***thay đổi giá trị tham số hình thức*** sẽ ***không ảnh hưởng tới biến được truyền***.
- **Truyền theo biến (tham chiếu)**
  - Tham số được truyền sẽ ***thực sự là biến*** và các thao tác sẽ thi hành trực tiếp với biến.
  - Những câu lệnh thay đổi giá trị tham số hình thức sẽ ***ảnh hưởng*** tới biến được truyền.

Truyền theo biến – Hàm nhận tham số là con trỏ

- Thực chất là truyền theo địa chỉ của biến
- Khai báo hàm [tham số phải chứa "***địa chỉ***"]: Khai báo là một con trỏ, trỏ tới một đối tượng có kiểu muốn truyền vào. ***Ví dụ:***

```
void Swap(int *X, int *Y) {
    int Temp = *X;
    *X = *Y;
    *Y = Temp;}
```

- **Truyền tham số**
  - Địa chỉ của biến được truyền. ***Ví dụ:*** `swap(&A,&B)`

Ví dụ

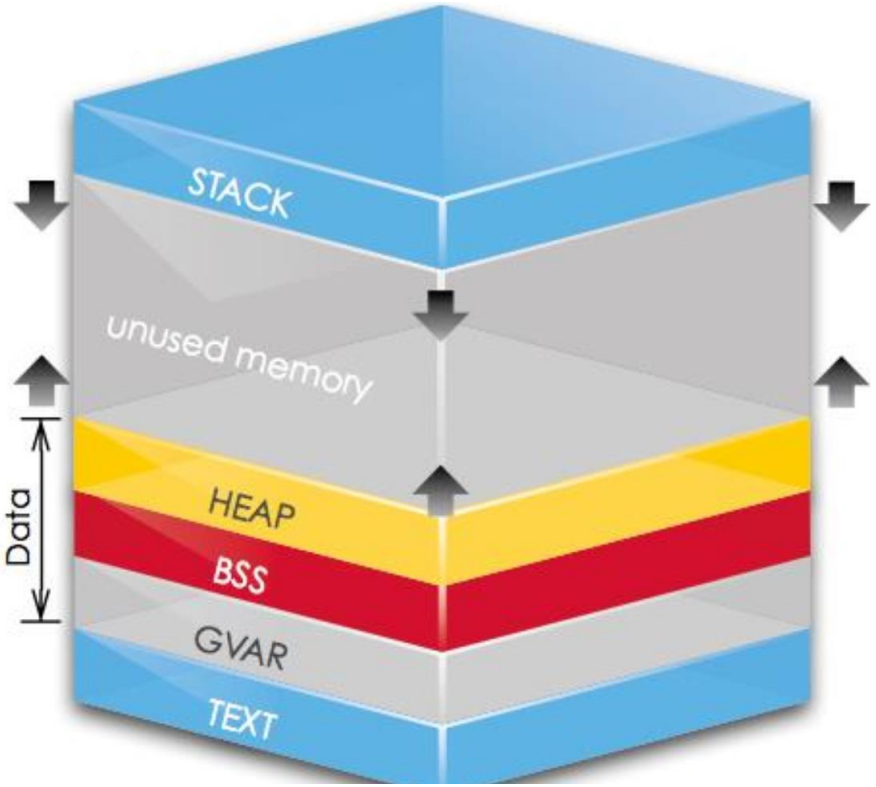
```c
# include <stdio.h>
# include <conio.h>
void swap(int * pa, int * pb) {
    int x = *pa;
    *pa = *pb;
    *pb = x;
    return;
}
void main(){
    int a = 5, b = 100;
    printf("Truoc: a=%d, b=%d \n\n",a,b);
    swap(&a,&b);
    printf("Sau : a=%d, b=%d\n\n",a,b);
    return;
}
```

```
Truoc: a=5, b=100

Sau : a=100, b=5
```

Con trỏ hàm

```
int foo(){
    return 0;
}
int main(){
    int n = foo();
    return 0;
}
```



- Khi trong hàm main chạy đến dòng lệnh gọi hàm foo, hệ điều hành sẽ tìm đến địa chỉ của hàm **`foo`** trên bộ nhớ ảo và chuyển mã lệnh của hàm **`foo`** cho CPU tiếp tục xử lý

Cú pháp khai báo con trỏ hàm

<return_type> (*<name_of_ptr>)(<data_type_of_parameters>);

- **Ví dụ 1:**

```
int foo(){
     return 0; }
int (*pFoo) ();
```

- **Ví dụ 2:**

```
void swapValue(int &value1, int &value2) {
     int temp = value1;
     value1 = value2;
     value2 = temp; }
void(*pSwap) (int &, int &);
```

Ví dụ sử dụng con trỏ hàm

```
void swapValue(int &value1, int &value2){
    int temp = value1;
    value1 = value2;
    value2 = temp;
}
int main(){
    void(*pSwap) (int &, int &) = swapValue;
    int a = 1, b = 5;
    cout << "Before: " << a << " " << b << endl;
    (*pSwap)(a, b);
    cout << "After: " << a << " " << b << endl;
    return 0;
}
```

Truyền theo biến – Hàm nhận tham số là tham chiếu

- Hàm nhận tham số là tham chiếu

```
void Swap(int &X, int &Y){
    int Temp = X;
    X = Y;
    Y = Temp; }
```

- Để hoán đổi giá trị hai biến A và B

```
Swap(A, B);
```

Câu hỏi 1: Kết quả đưa ra màn hình?

```c
#include<stdio.h>
void fun(int n){
    if(n > 0) {
        fun(--n);
        printf("%d ", n);
        fun(--n);
    }
}
int main(){
    fun(3);
    return 0;
}
```

A	0 2 1 0
B	0 1 0 2
C	1 1 2 0
D	0 1 2 0
E	0 2 0 1

Truyền tham chiếu

- Khi một hàm trả về một tham chiếu, chúng ta có thể gọi hàm ở phía bên trái của một phép gán

```
#include <iostream.h>
int X = 4;
int & MyFunc(){
    return X;
}
int main(){
    Cout << "X=" << X << endl;
    Cout << "X=" << MyFunc() << endl;
    MyFunc() = 20; // ~X=20
    Cout << "X=" << X << endl;
    return 0;
}
```

Tham số ngầm định (1/2)

- Định nghĩa các giá trị tham số mặc định cho các hàm
- Ví dụ

```
void MyDelay(long Loops = 1000){
    for(int I = 0; I < Loops; ++I) ;
}
```

- `MyDelay();` // Loops có giá trị là 1000
- `MyDelay(5000);` // Loops có giá trị là 5000

Tham số ngầm định (2/2)

- Nếu có prototype, các tham số có giá trị mặc định chỉ được cho trong prototype của hàm và không được lặp lại trong định nghĩa hàm.
- Một hàm có thể có nhiều tham số có giá trị mặc định. Các tham số có giá trị mặc định cần phải được nhóm lại vào các tham số cuối cùng (hoặc duy nhất) của một hàm. Khi gọi hàm có nhiều tham số có giá trị mặc định, chúng ta chỉ có thể bỏ bớt các tham số theo thứ tự từ phải sang trái và phải bỏ liên tiếp nhau
- Ví dụ:

```
int MyFunc(int a = 1, int b, int c = 3, int d = 4); // ✗
int MyFunc(int a, int b = 2, int c = 3, int d = 4); // ✓
```

Đa năng hóa hàm (Overloading)

- Cung cấp nhiều hơn một định nghĩa cho tên hàm đã cho trong cùng một phạm vi.
- Trình biên dịch sẽ lựa chọn phiên bản thích hợp của hàm hay toán tử dựa trên các tham số mà nó được gọi.

C

```
int abs(int i);
long labs(long l);
double fabs(double d);
```

C++

```
int abs(int i);
long abs(long l);
double abs(double d);
```

Từ khóa auto (1/2)

- Đối với biến (từ C++11): **auto** xác định kiểu của biến được khởi tạo một cách tự động từ giá trị khởi tạo của biến.

```
auto d { 5.0 }; // d will be type double
auto i { 1 + 2 }; // i will be type int
```

- • Đối với hàm (từ C++14): **auto** tự động xác định kiểu trả về của hàm dựa vào câu lệnh return.

```
auto add(int x, int y) -> int;
auto divide(double x, double y) -> double;
auto printSomething() -> void;
auto generateSubstring(const std::string
        &s, int start, int len) -> std::string;
```

Từ khóa auto (2/2)

- Đối với kiểu tham số (từ C++14): auto tự động xác định kiểu của tham số dựa vào giá trị được truyền. Ví dụ:

```
auto maxval(auto x, auto y){
    return (x > y) ? x : y; }
int main() {
    int i = maxval(3, 7); // returns 7
    cout << i << endl;
    double d = maxval(6.34, 18.523); // returns 18.523
    cout << d << endl;
    char ch = maxval('a', '6'); // returns 'a'
    cout << ch << endl;
    return 0; }
```

Câu hỏi 2: Kết quả đưa ra màn hình?

```c
#include<stdio.h>
void fun(int *i, int *j){
    *i = *i * *i;
    *j = *j * *j;
}
int main(){
    int i=5, j=2;
    fun(&i, &j);
    printf("%d, %d", i, j);
    return 0;
}
```

A	5, 2
B	2, 5
C	10, 4
D	4, 25
E	25, 4

Câu hỏi 3: Kết quả đưa ra màn hình?

```c
#include<stdio.h>
void fun(char *a){
    printf("%c", *++a);
    a++;
    printf("%c", *a);
}
int main(){
    void fun(char*);
    char a[10]="ABCDEF";
    fun(&a[0]);
    return 0;
}
```

A	AB
B	AC
C	BC
D	BD
E	CD

Hàm Đệ Quy

Đệ quy là gì (Recursion) (1/2)

- Định nghĩa tường minh: Giải thích khái niệm mới bằng những khái niệm đã có.
  - Người = Động vật bậc cao.
- Định nghĩa khác: Giải thích một khái niệm bằng chính khái niệm đó.
- Đệ quy: Đưa ra một định nghĩa có sử dụng khái niệm đang cần định nghĩa (quay về).
  - Ví dụ: Người = con của người khác.
  - Ví dụ: A sinh ra B, B sinh ra C. **Đệ quy:** C là con của con A.

Đệ quy là gì (Recursion) (2/2)

- Con người hiểu được định nghĩa đệ quy vì đệ quy có "chặn" (điều kiện biên, điều kiện suy biến) – Có thể là biên ngầm định
  - Người = con của hai người khác

    => Ngầm hiểu là có 2 người đầu tiên.
  - Thư mục = các thư mục con + các tập tin.

    => Ngầm hiểu: Hiển nhiên tồn tại thư mục gốc là cả ổ đĩa

Tìm giải thuật đệ quy (1/3)

1) Thông số hóa bài toán

- Tổng quát hóa bài toán cụ thể cần giải thành bài toán tổng quát (một họ các bài toán chứa bài toán cần giải)
- Tìm ra các thông số cho bài toán tổng quát
  - các thông số điều khiển: các thông số mà độ lớn của chúng đặc trưng cho độ phức tạp của bài toán , và giảm đi qua mỗi lần gọi đệ quy.
    - **Ví dụ:**
    - `n trong hàm FAC(n) ;`
    - `a , b trong hàm USCLN(a,b) .`

Tìm giải thuật đệ quy (2/3)

2) Tìm các trường hợp neo cùng giải thuật giải tương ứng

- trường hợp suy biến của bài toán tổng quát
- các trường hợp tương ứng với các gía trị biên của các biến điều
- khiển

Ví Dụ: `FAC(1) =1`

`USCLN(a,0) = a`

Tìm giải thuật đệ quy (3/3)

3) Tìm giải thuật giải trong trường hợp tổng quát bằng phân rã bài toán theo kiểu đệ quy

- Tìm phương án (giải thuật) giải bài toán trong trường hợp tổng quát phân chia nó thành các thành phần
  - giải thuật không đệ quy
  - bài toán trên nhưng có kích thước nhỏ hơn.
- Ví dụ

```
FAC(n) = n * FAC(n -1) .
Tmax(a[1:n]) = max(Tmax(a[1:(n-1)]) , a[n] )
```

Phân tích các hàm đệ quy (1/2)

- **Ví dụ: Hàm tính giai thừa**

```
int Fact(int n)
{
      if (n <= 1)
            return  1;
      else return n * Fact(n-1);
}
```

Với n <= 1, chương trình chỉ thực hiện so sánh và thực hiện 1 lệnh return 1

Với n > 1, chương trình phải gọi đệ quy **fact(n-1)**, việc gọi đệ quy này tốn **T(n-1)**, sau khi có kết quả của việc gọi đệ quy, chương trình phải nhân kết quả đó với n và return tích số (tốn O(1)), tức là T(n) = 0(1) + T(n-1)

- Với n <= 1, ta có T(1) = O(1).
- Với n > 1, ta có T(n) = 0(1) + T(n-1)

Phân tích các hàm đệ quy (2/2)

- Ta có quan hệ đệ quy sau:
  - $T(1) = O(1)$
  - $T(n) = T(n-1) + O(1)$ với $n > 1$
- Thay các ký hiệu O(1) bởi các hằng số dương a và b tương ứng, ta có
  - $T(1) = a$
  - $T(n) = T(n-1) + b$ với $n > 1$

Hàm Đệ Quy

- Quá trình trong đó một hàm gọi lại chính nó một cách liên tiếp được gọi là đệ quy.
- Sử dụng đệ quy làm code trở nên dễ đọc, chặt chẽ nhưng lại khó hiểu

```
kieu_tra_ve tenhamdequi() {
    tenhamdequi(); /* goi lai chinh no */
}

int main() {
    tenhamdequi();
}
```

Ví dụ: Tính giai thừa

```c
#include <stdio.h>
int GiaiThua(int n) {
    if (n == 1)
        return 1;
    else
        return (n * GiaiThua(n - 1));
}
int main() {
    printf("Giai thua cua 5 la: %d", GiaiThua(5));
    return 0;
}
```

```
Giai thua cua 5 la: 120
```

Cách thức hoạt động – Giai thừa

```
GiaiThua(5)
   GiaiThua(4)
      GiaiThua(3)
         GiaiThua(2)
            GiaiThua(1)
               return 1
            return 2*1 = 2
         return 3*2 = 6
      return 4*6 = 24
   return 5*24 = 120
```

Ví dụ: Dãy số Fibonacci

```c
#include <stdio.h>
static int n1 = 0, n2 = 1, n3 = 0;
void printFibo(int count) {
    if (count > 0) {
        n3 = n1 + n2;
        n1 = n2;
        n2 = n3;
        printf(" %d", n3);
        printFibo(count - 1);
    }
}
int main() {
    int count = 15;
    printf("%d %d", n1, n2); // in 0 và 1
    printFibo(count - 2); // n-2 vì 2 so 0 và 1 da duoc in ra
    return 0;}
```

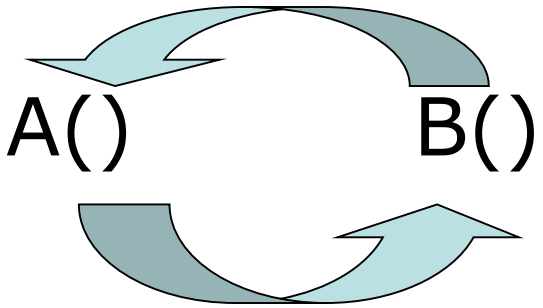
```
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

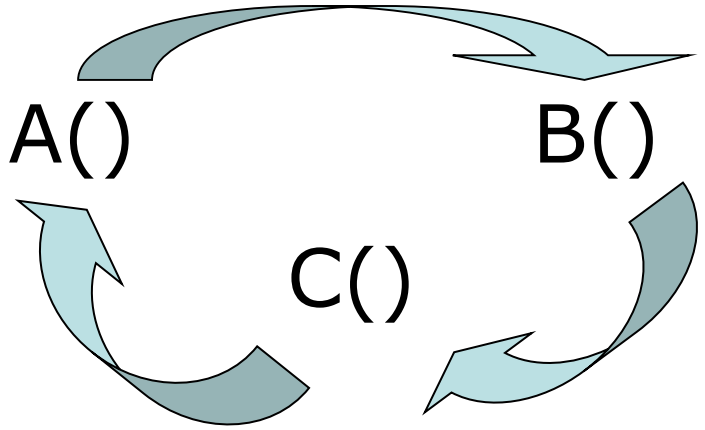
Phân loại giải thuật đệ qui

- Đệ quy phân thành 2 loại :
- Đệ quy trực tiếp:
- Đệ quy gián tiếp (Tương hỗ):

❖Đệ quy trực tiếp: Hàm chứa lời gọi đến chính nó

❖Đệ quy gián tiếp (Tương hỗ): Hàm chứa lời gọi đến hàm khác, mà ở hàm này lại chứa lời gọi đến chính nó





Cài đặt hàm đệ quy (1/2)

- Hàm đệ quy về cơ bản gồm hai phần:
  1) **Phần cơ sở (Phần neo):** chứa các tác động của hàm với một số giá trị suy biến (neo) của tham số.
  2) **Phần đệ quy:** Chứa các lời gọi đệ quy giải quyết các vấn đề con với cỡ nhỏ hơn. Lời gọi đệ qui dần tiến tới trường hợp neo

Cài đặt hàm đệ quy (2/2)

- Cấu trúc hàm đệ qui như sau

```
if (suy biến)
    <Giải quyết trường hợp suy
  biến>;
else {
    <tiền xử lý đệ qui>;
    <Gọi đệ qui> ;
    <Xử lý hậu đệ qui>;
}
```

Chú ý: những lệnh sau lời gọi đệ qui được lưu vào Stack.

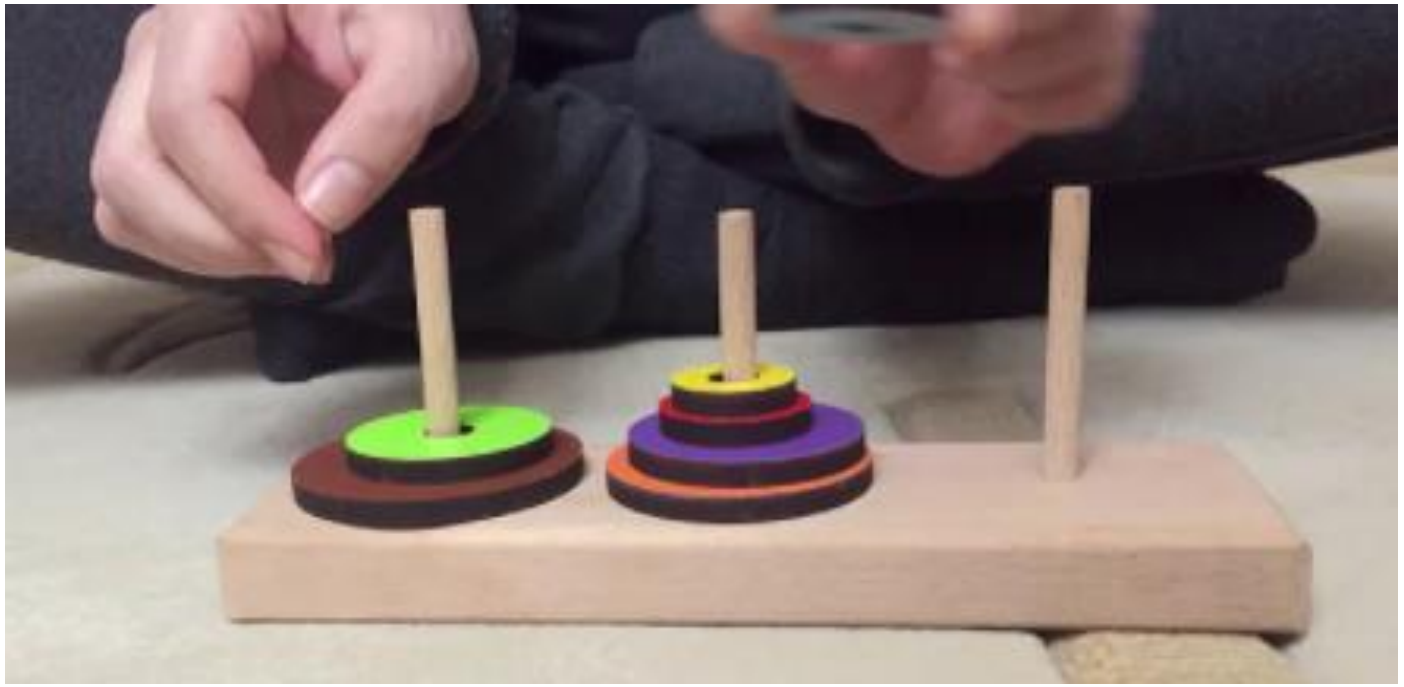
Thiết kế giải thuật đệ qui

- Để xây dựng giải thuật đệ quy, ta cần thực hiện tuần tự 3 nội dung sau :
  - Thông số hóa bài toán.
  - Tìm các trường hợp neo cùng giải thuật giải tương ứng.
  - Tìm giải thuật giải trong trường hợp tổng quát bằng phân rã bài toán theo kiểu đệ quy.

Ưu và nhược điểm của đệ qui

- Ưu điểm của đệ quy
  - Sáng sủa, dễ hiểu, nêu rõ bản chất vấn đề
  - Tiết kiệm thời gian hiện thực mã nguồn
- Nhược điểm của đệ quy
  - Tốn nhiều bộ nhớ, thời gian thực thi lâu
  - Một số bài toán không có lời giải đệ quy

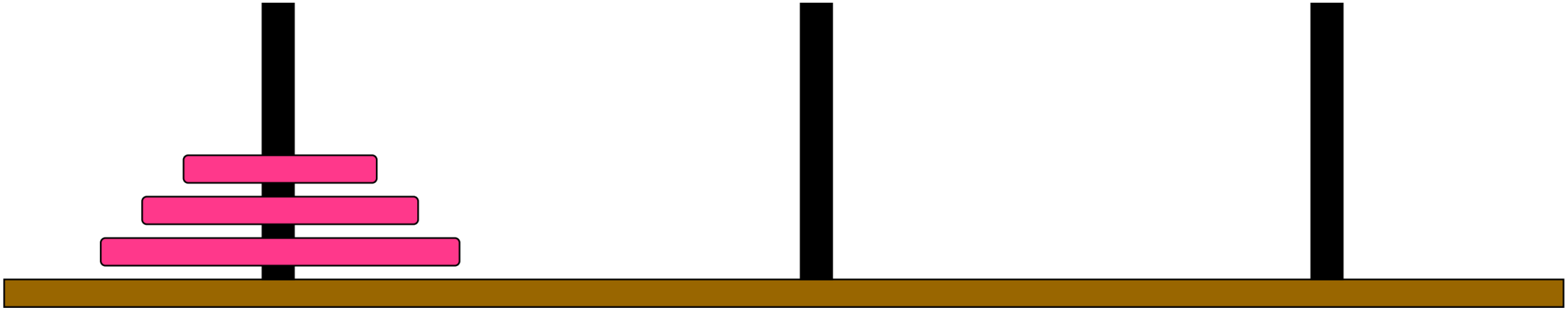
Trò chơi tháp Hà Nội



https://www.youtube.com/watch?v=h25dsNF6564

Bài toán Tháp Hà Nội (1/3)

- **Luật**:
  - Di chuyển mỗi lần một đĩa
  - Không được đặt đĩa lớn lên trên đĩa nhỏ



Với chồng gồm n đĩa cần 2n-1 lần chuyển

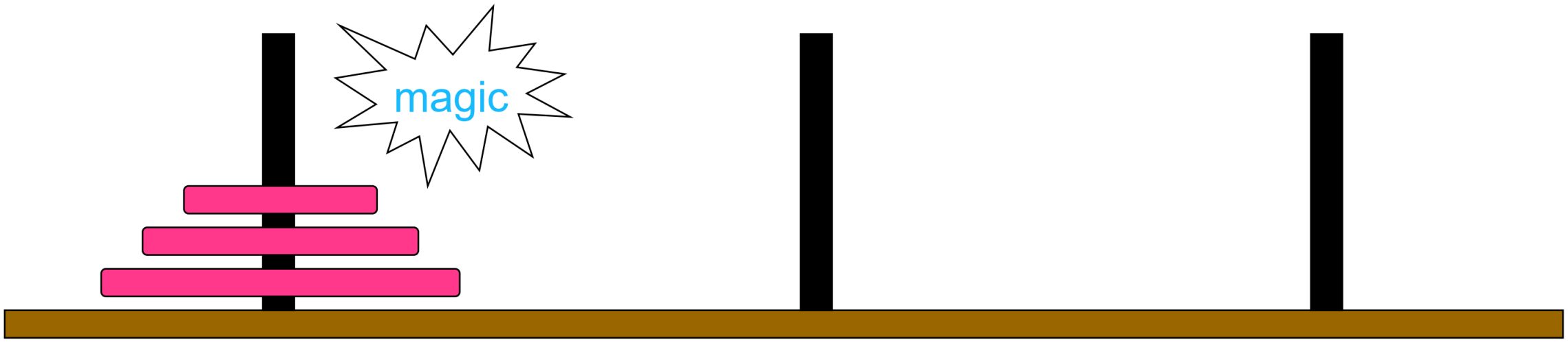
–Giả sử thời gian để chuyển 1 đĩa là t giây thì thời gian để chuyển xong chồng 64 đĩa sẽ là:

*–T = (2^64-1) * t = 1.84 * 10^19 t*

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

Bài toán Tháp Hà Nội (2/3)

- **Hàm đệ quy**: Chuyển n đĩa từ A sang C qua trung gian B
  - Chuyển n-1 đĩa trên đỉnh của cột A sang cột B
  - Chuyển 1 đĩa (cuối cùng) của cột A sang cột C
  - Chuyển n-1 đĩa từ cột B sang C qua trung gian A



Bài toán Tháp Hà Nội (3/3)

- Thông số hóa bài toán
  - Xét bài toán ở mức tổng quát nhất: chuyển n (n>=0) đĩa từ cột A sang cột C lấy cột B làm trung gian .
  - `THN(n,A,B,C)` -> với 64 đĩa gọi `THN(64,A,B,C)`
  - n sẽ là thông số quyết định bài toán –n là tham số điều khiển
- ▪ Trường hợp suy biến và cách giải
  - Với n =1 : `THN (1,A,B,C)`
- Giải thuật giải bt THN (1,A,B,C) là thực hiện chỉ 1 thao tác cơ bản: Chuyển 1 đĩa từ A sang C (ký hiệu là Move (A , C))
  - `THN(1,A,B,C) ≡ { Move( A, C ) }`
  - `THN(0,A,B,C) ≡ { φ}`

Giải thuật tổng quát Tháp Hà Nội

Với n>1

```
THN(n,A,B,C) ≡ {
   THN (n -1,A,C,B);
   Move ( A, C );
   THN (n -1,B,A,C);
}
```

```
void move(int n, int A, int B, int C){
      if (n > 0) {
      move(n - 1, A, C, B);
      printf("\n Move disk % d from %c to % c ", n, A,C );
      move(n - 1, B, A, C);
      }
}
```

Ví dụ

- **Tính: S(n) = 1/(1*2) + 1/(2*3) + ... + 1/(n*(n+1))**

S(n) = 1/2 (khi n==1)

= S(n-1)+1/(n*(n+1))

```
float S(int n) {
    if ( n==1) return 0.5;
    else return S(n-1)+1.0/(n*(n+1));
}
```

CHƯƠNG 4
MẢNG & THAO TÁC TRÊN MẢNG

GIẢNG VIÊN

TS. Hà Ngọc Long

Khái niệm mảng (1/2)

- Trong thực tế, thường gặp các đối tượng có tính chất chung
  - Tháng trong năm
  - Điểm trung bình của sinh viên trong lớp
- Các đối tượng được nhóm lại dưới một tên
- Đối tượng được đặc trưng bởi tên nhóm và thứ tự trong nhóm
  - Tháng thứ 3 trong năm: Tháng 3
  - Sinh viên thứ 17 trong lớp:…
- Số thứ tự trong nhóm là chỉ số phần tử

Khái niệm mảng (2/2)

- Kiểu mảng là một kiểu dữ liệu gồm
  - Một số hữu hạn *thành phần*.
  - Các thành phần có *cùng một kiểu*: kiểu cơ sở hay là kiểu thành phần.
- Mỗi phần tử của mảng được tham khảo thông qua:
  - Tên mảng và,
  - Chỉ số của phần tử trong mảng.

Khai báo mảng

```
Kiểu_dữ_liệu Tên_Mảng[Kích thước];
```

- **Kiểu_dữ_liệu:** kiểu của các phần tử trong mảng (nguyên, thực, ký tự, chuỗi, mảng,…)
  - **Tên_mảng**: tên của mảng
  - **Kích_thước_mảng**: số phần tử trong mảng
- **Ví dụ**

```
// khai báo mảng 10 phần tử có kiểu dữ liệu int
int Mang_so_nguyen[10];
float A[10]; //⇐ Mảng 10 phần tử kiểu số thực
```

Cấp pháp bộ nhớ cho mảng

- Các phần tử trong mảng được cấp phát các ô nhớ kế tiếp nhau trong bộ nhớ
- Kích thước của mảng bằng kích thước một phần tử nhân với số phần tử

❑**Ví dụ:**

```
int A[10]; //Mảng A gồm 10 phần tử nguyên
```

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]

Kích thước của mảng A: 10 x 2 = 20 bytes

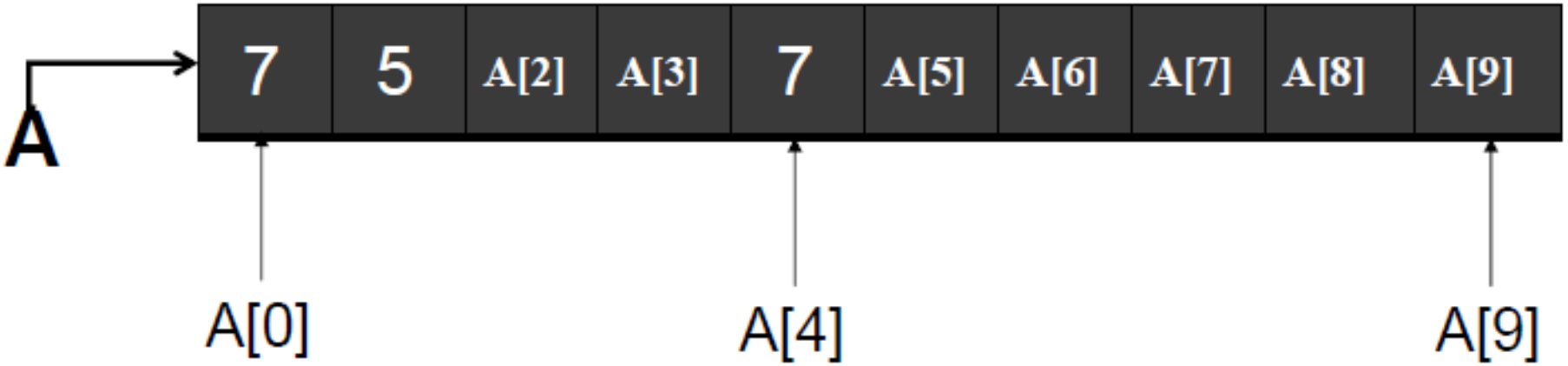
Truy cập đến thành phần của mảng

- Biến mảng lưu trữ địa chỉ ô nhớ đầu tiên trong vùng nhớ được cấp phát
- Ngôn ngữ C đánh chỉ số các phần tử trong mảng bắt đầu từ 0
- Các phần tử của mảng được truy nhập thông qua
  - Tên mảng và
  - Chỉ số của phần tử của phần tử trong mảng

```
Tên_Mảng[Chỉ_số_phần_tử];
```

Ví dụ 1

```
int A[10];
```
//Mảng A gồm 10 phần tử nguyên




```
A[0] = 7;

A[1] = 5;

A[4] = 7;

int N = A[1] + A[4]; → N = 14
```

Ví dụ 2

```
int A[10];
for(int i = 0; i < 10; i++) A[i]= 2* i;
```

```
0 2 4 6 8 10 12 14 16 18
```

- **Chú ý:** C không kiểm tra vượt quá giới hạn của mảng khi truy nhập

```
int A[3], B[4], C[3];
```

```
A[0] A[1] A[2] B[0] B[1] B[2] B[3] C[0] C[1] C[2]
```

`A[5]` ⇔ `B[2]` ⇔ `C[-2]` ←nếu cấp phát vùng nhớ liên tiếp

Mảng nhiều chiều

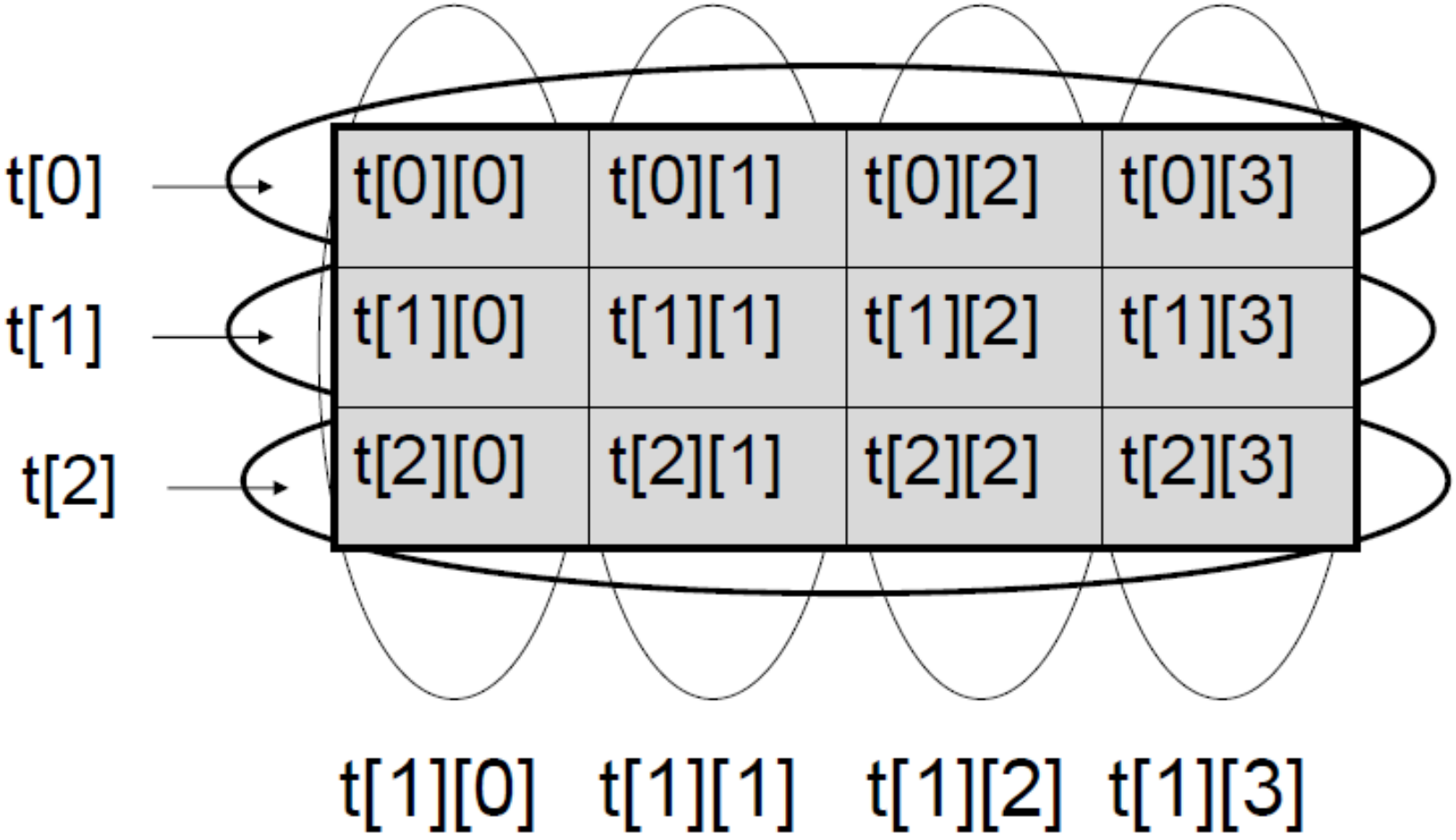
- Mỗi phần tử của mảng có thể là một mảng
- Mảng nhiều chiều

```
Kiểu Tên[Chiều_1] [Chiều_2]… [Chiều_N];
```

- **Kiểu**: Kiểu của mỗi phần tử trong mảng
- **Chiều_1**, **Chiều_2**,…**Chiều_N**: Các hằng số nguyên, cho biết kích thước (số phần tử) của mỗi chiều
- Mảng gồm: **Chiều_1** x **Chiều_2** x...x **Chiều_N** phần tử được lưu trữ trong vùng nhớ liên tục. Các phần tử thuộc kiểu **Kiểu**

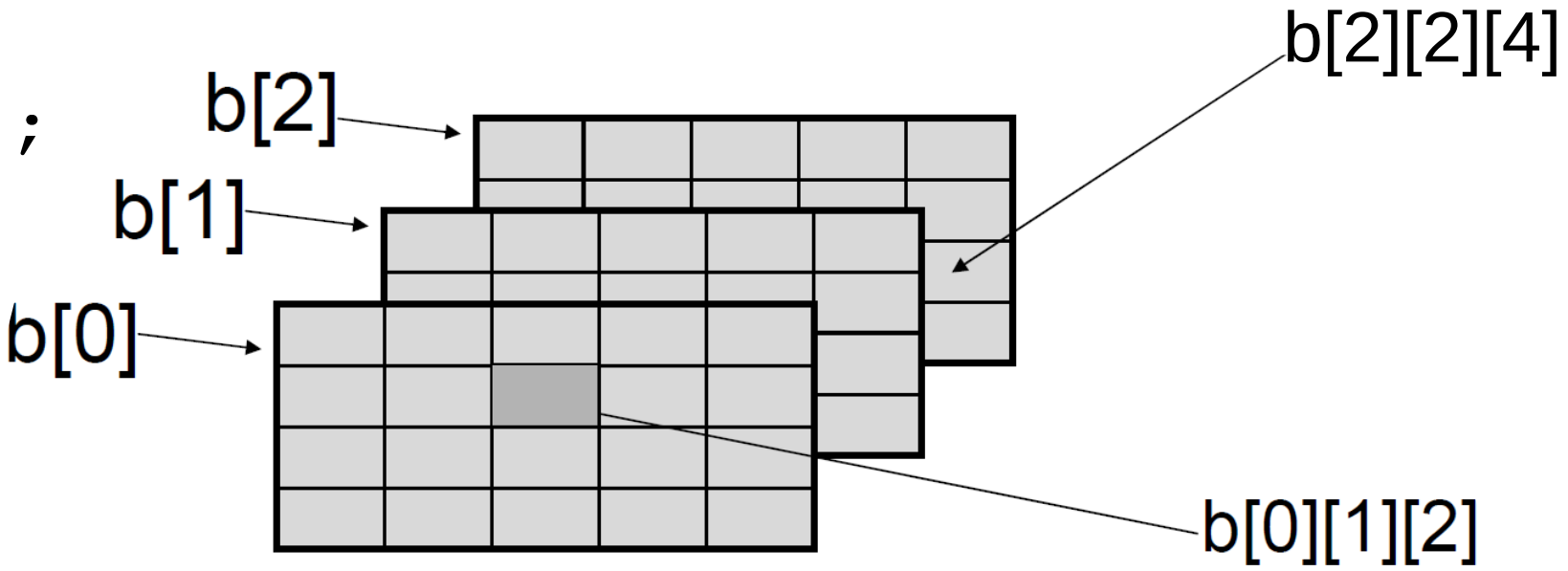
Mảng nhiều chiều

```
int t[3][4];
```



Ví dụ

```
int b[3][4][5];
```



- Mảng b gồm 3 phần tử `b[0], b[1], b[2]`
- Mỗi phần tử là mảng hai chiều gồm 4 hang (hàng 0, 1, 2, 3) và 5 cột (0, 1, 2, 3, 4)
- Mỗi phần tử có kiểu nguyên có dấu, 2 byte

Khởi tạo giá trị cho mảng

- Các phần tử của mảng có thể được khởi tạo giá trị ngay khi khai báo
- **Ví dụ**

```
int a[4] = {1,4,6,2};
int b[2][3]={ {1,2,3}, {4,5,6} };
int t[3][4] = {
                  {1, 2, 3, 4},
                  {5, 6, 7, 8},
                  {9, 10, 11, 12},
                };
```

Chú ý

- Số lượng giá trị khởi tạo không được lớn hơn số lượng phần tử của mảng
  - Nếu số lượng này nhỏ hơn, các phần tử còn lại được khởi tạo giá trị 0

```
int A[3][4] = { {1}, {4,5} };
```

```
int A[3][4] = { }; ← Tất cả đều mang giá trị 0
```

- Có thể xác định kích thước mảng thông qua số giá trị khởi tạo nếu để trống kích thước mảng

```
int A1[8] = {2, 4, 6, 8, 10, 12, 14, 16};
```

```
int A2[] = {2, 4, 6, 8, 10, 12, 14, 16};
```

Các thao tác thường gặp trong mảng

- **Nhập/Xuất dữ liệu cho mảng**
  - Mảng 1 chiều, ma trận
- **Bài toán đếm**
  - Đếm số phần tử
  - Tính toán trên các phần tử..
- **Tìm kiếm phần tử**
  - Lớn nhất/nhỏ nhất/bất kỳ
- **Sắp xếp phần tử trong mảng**
  - Theo thứ tự, theo nguyên tắc
- **Chèn thêm phần tử, xóa phần tử**

Nhập dữ liệu

Dùng hàm **scanf()**

- **Ví dụ:** `int Tab[10];`
- Nhập dữ liệu cho một phần tử

  `scanf("%d",&Tab[2]);` ←phần tử thứ 3 của mảng
- Nhập dữ liệu cho cả mảng
  - Dùng vòng lặp **for**

```
for(i = 0; i < 10; i++)
    scanf("%d",&Tab[i]);
```

  - Nên in ra chỉ số phần tử khi nhập

```
printf("Tab[%d] : "); scanf("%d",&Tab[i])
```

Ví dụ 1

Nhập vào lượng mưa (mm) trong năm

```
Nhap luong mưa tháng 1: 100
Nhap luong mưa tháng 2: 100
Nhap luong mưa tháng 3: 15
Nhap luong mưa tháng 4: 123
Nhap luong mưa tháng 5: 4213
Nhap luong mưa tháng 6: 123
Nhap luong mưa tháng 7: 123
Nhap luong mưa tháng 8: 4
Nhap luong mưa tháng 9: 5
Nhap luong mưa tháng 10: 6
Nhap luong mưa tháng 11: 7
Nhap luong mưa tháng 12: 5
```

```c
#include <stdio.h>
#define MONTHS 12
int main(){
    int rainfall[MONTHS], i;
    for ( i=0; i < MONTHS; i++ ){
        printf("Nhap luong mưa tháng %d: ", i+1);
        scanf("%d", &rainfall[i] );
    }
    return 0;
}
```

Lưu ý

- Nếu số phần tử của mảng chỉ được biết tại thời điểm thực hiện chương trình (nhưng biết số phần tử tối đa)
  - Khai báo mảng với kích thước tối đa
  - Sử dụng biến nguyên lưu số phần tử thực sự của mảng.
- **Ví dụ:** Nhập vào mảng không quá 100 số thực
  - Khai báo mảng thực Tab có tối đa 100 phần tử.
  - Nhập số phần tử thực sự của mảng
  - Nhập giá trị cho từng phần phần tử (dung vòng lặp **`for`**)

Ví dụ 2

```c
#include<stdio.h>
void main(){
    float A[100];
    int n, i;
    do {
        printf("\n Cho biet so phan tu cua mang: ");
        scanf("%d",&n);
    } while (n>100 || n<=0);
    for(i = 0; i < n; i++){
        printf("A[%d] = ", i); scanf("%f",&A[i]);
    }
}
```

```
 Cho biet so phan tu cua mang: 5
A[0] = 1
A[1] = 2
A[2] = 3
A[3] = 4
A[4] = 5
```

Xuất dữ liệu trong mảng

Dùng hàm printf()

❑ ***Ví dụ:*** `int Tab[10];`

- Hiện thị phần tử thứ 5:

```
printf("%d",Tab[4]);
```

- Để hiển thị tất cả các phần tử:

```
for(i = 0; i < 10; i++)
    printf("%4d",Tab[i]);
```

- **Các kiểu xuất dữ liệu**
  - Hiển thị tất cả/một phần theo dòng/cột..
  - Hiển thị từng k phần tử trên một dòng…

Ví dụ 1

```c
#include <stdio.h>
#define MAX 12
void main(){
    int A[MAX], i;
    for (i=0; i < MAX; i++){ //Nhập dữ liệu
        printf("A[%d]: ", i+1); scanf("%d", &A [i] );
    }
    for (i =0; i < MAX; i++){ //Xuất dữ liệu
        printf( "%4d" , A[i]);
        if( (i+1) %4==0) printf("\n");
    }
}
```

```
A[1]: 12
A[2]: 13
A[3]: 14
A[4]: 15
A[5]: 16
A[6]: 17
A[7]: 18
A[8]: 19
A[9]: 20
A[10]: 21
A[11]: 22
A[12]: 23
  12  13  14  15
  16  17  18  19
  20  21  22  23
```

Ví dụ 2

```c
#include <stdio.h>
void main(){
    int A[20][20], n, m, i,j;
    printf("Nhap so hang : "); scanf("%d",&n);
    printf("Nhap so cot : "); scanf("%d",&m);
    printf("\n");
    for ( i=0; i < n; i++ )
        for(j=0; j < m; j++) {
        printf("Nhap phan tu A[%d,%d]: ", i+1,j+1);
        scanf("%d", &A[i][j] );
        }
    printf("\n\n MA TRAN DA NHAP \n\n");
    for ( i=0; i < n; i++ ){
        for(j=0; j < m; j++)
            printf( "%4d" ,A[i][j]);
        printf("\n");
    }
}
```

```
Nhap so hang : 2
Nhap so cot : 3

Nhap phan tu A[1,1]: 1
Nhap phan tu A[1,2]: 2
Nhap phan tu A[1,3]: 3
Nhap phan tu A[2,1]: 4
Nhap phan tu A[2,2]: 5
Nhap phan tu A[2,3]: 6


 MA TRAN DA NHAP

   1   2   3
   4   5   6
```

Đếm số phần tử thỏa mãn điều kiện

- Duyệt từng phần tử của dãy (dùng for)
- Nếu phần tử xét thỏa mãn điều kiện
  - Ghi nhận
- Chuyển sang xem xét phần tử tiếp theo
- **Ví dụ:** Đếm số tháng có lượng mưa lớn hơn 50mm

```
int dem = 0;
for(i = 0; i < MONTHS; i++)
    if(rainfall[i] > 50)
        dem++;
printf("\nThang mua nhieu hon 50mm: %d", dem);
```

Ví dụ: Nhập mảng, đưa ra TBC các số chia hết 7

```c
#include<stdio.h>
void main(){
    int A[100];
    int n, i, d = 0, S=0;
    printf("\n So phan tu cua mang (<100) : "); scanf("%d",&n);
    for(i = 0; i < n; i++){
        printf("A[%d] = ", i); scanf("%d",&A[i]);
    }
    for(i = 0; i < n; i++)
        if(A[i] %7==0){
            d++;
            S+= A[i];
        }
    if(d > 0) printf("TBC so chia het cho 7: %7.2f",(float)S/d);
    else printf("Trong day khong co so chia het cho 7");
}
```

```
 So phan tu cua mang (<100) : 5
A[0] = 10
A[1] = 7
A[2] = 14
A[3] = 21
A[4] = 28
TBC so chia het cho 7:   17.50
```

Tìm kiếm phần tử (1/4)

- Tìm phần tử lớn nhất (nhỏ nhất)
  - Giả sử phần tử đó là phần tử đầu tiên
  - Lần lượt so sánh với các phần tử còn lại
    - Nếu phần tử mới của dãy lớn hơn ⇒ coi đây là phần tử lớn nhất và tiếp tục so sánh với phần tử kế
    - Nếu không đúng, so sánh tiếp với phần tử kế

Tìm kiếm phần tử (2/4)

- Ví dụ: Tìm tháng có lượng mưa nhiều nhất trong năm

```
max = rainfall[0];
for(i = 1; i < MONTHS; i++)
    if(rainfall[i] > max)
        max = rainfall[i];
printf("\n Luong mua nhieu nhat la: %d", max);
```

Tìm kiếm phần tử (3/4)

- Tìm kiếm các phần tử thỏa mãn điều kiện (giống bài toán đếm)
  - Dùng for duyệt toàn bộ
  - Nếu cần thiết, dùng thêm mảng ghi lại chỉ số
- **Ví dụ:** Đưa ra danh sách các tháng có lượng mưa nhiều hơn 500mm

```
printf("Thang co luong mua lon hon 500mm")
for(i = 0; i < MONTHS; i++)
    if(rainfall[i] > 500)
        printf("\nThang %d", i+1);
```

Tìm kiếm phần tử (4/4)

- Tìm phần tử đầu tiên của danh sách
  - Dùng vòng lặp for kết hợp với break;
  - Dùng vòng lặp while

- **Ví dụ:** Đưa ra phần tử đầu của mảng có giá trị bằng k;

Ví dụ

```
int Tab[100]
int N, i, k, f;//N: số phần tử, k phần tử cần tìm
```

Dùng for

```
for(i = 0; i < N; i++)
    if(Tab[i] == k) break;
if(i< N) printf("Tim thay tai vi tri %d",i);
```

Ví dụ

```
int Tab[100]
int N, i, k, f;//N: số phần tử, k phần tử cần tìm
```

Dùng while

```
i=0; f =0; //f: found. f = 1 ⇔ k is found
while(i < N && f==0){
    if(Tab[i] == k) f = 1;
    else i++;}
if (f==1) printf("Tim thay tai vi tri %d",i);
```

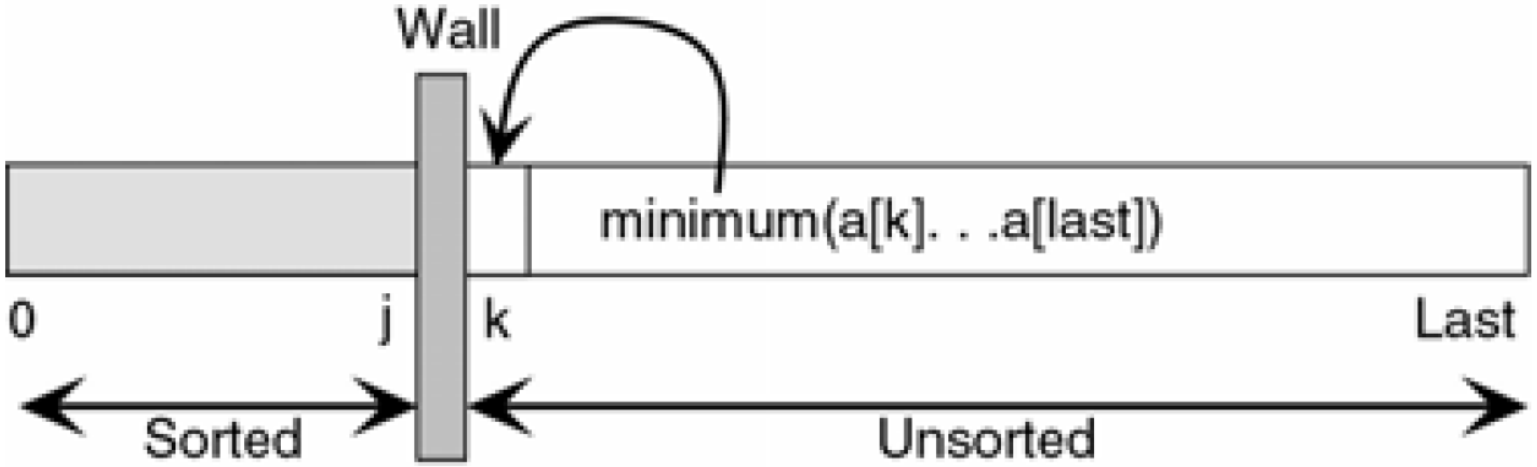
Bài toán sắp xếp theo thứ tự

- Cho mảng phần tử, sắp xếp theo thứ tự tăng/giảm
- **Các thuật toán**
  - Sắp xếp thêm dần (insertion sort)
  - Sắp xếp lựa chọn (selection sort)
  - Sắp xếp nổi bọt (bubble sort)
  - Sắp xếp vun đống (heap sort)
  - Sắp xếp nhanh (quick sort)
  - Sắp xếp trộn (merge sort)
  - ….

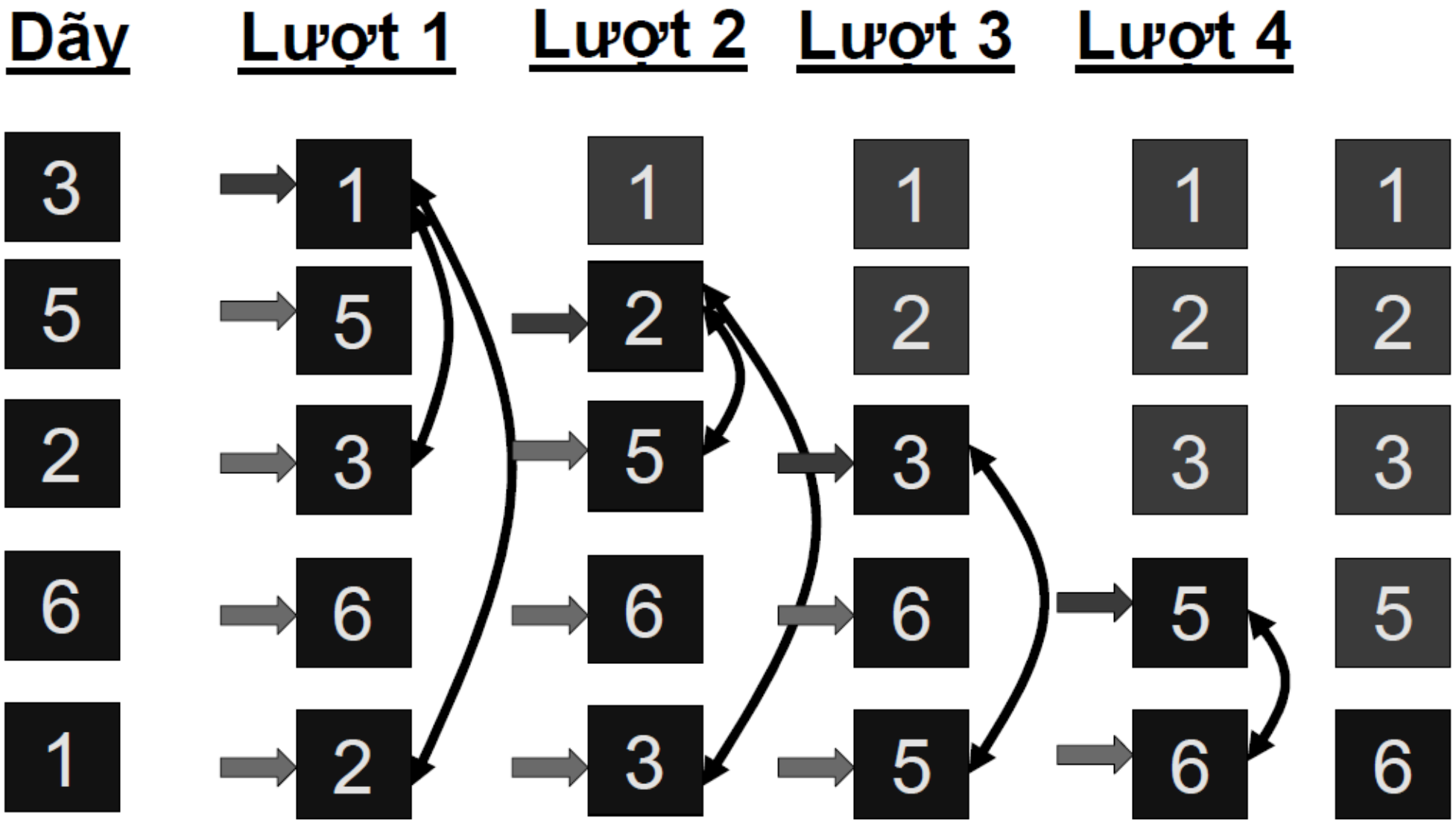
Thuật toán sắp xếp lựa chọn (Tăng) (1/3)

❑ Nguyên tắc

- Tại lượt sắp thứ k, tìm phần tử nhỏ nhất trong số các phần tử chưa được sắp xếp ([k..last]) và đổi chỗ cho phần tử thứ k (có chỉ số k-1)
  - Khi k = 1, phần tử thứ nhất (chỉ số 0) đúng vị trí
  - Khi k= 2, phần tử thứ hai (chỉ số 1) đúng vị trí…



Thuật toán sắp xếp lựa chọn (Tăng) (2/3)



Thuật toán sắp xếp lựa chọn (Tăng) (3/3)

//Khai báo các biến

```
int A[100]; //Mảng chứa dữ liệu
int N, i, j, tmp;
```

//Sắp xếp

```
for(i = 0; i < N - 1; i++)
  for(j = i + 1; j < N; j++)
      if(A[i] > A[j]) {
          tmp = A[i];
          A[i] = A[j];
          A[j] = tmp;
      }
```

Ví dụ

- Nhập vào từ bàn phím một mảng các số nguyên không quá 100 phần tử
- Hiển thị dãy số vừa nhập
- Sắp xếp dãy theo thứ tự giảm dần
  - Hiện thị dãy tại mỗi lượt sắp xếp

Ví dụ: Code

```c
#include<stdio.h>
void main(){
    int A[100] ;
    int N, i, j , t;
    printf("So phan tu [< 100]: "); scanf("%d",&N);
    printf("Hay nhap day so...\n");
    for(i=0; i < N; i++){
        printf("A[%d] = ",i+1); scanf("%d",&A[i]);
    }
    printf("\nDay vua nhap...\n");
    for(i=0; i < N; i++)
        printf("%4d", A[i]);
    printf("\n\n");
```

```c
printf("Sap xep day theo thuat toan
lua chon");
    for(i=0; i < N-1; i++){
        for(j=i; j < N; j++)
            if(A[i] < A[j]) {
                t = A[i];
                A[i] = A[j];
                A[j] = t;
            }//if & for_j
        printf("\nLuot %d : ",i+1);
        for(j=0; j < N; j++)
            printf("%4d", A[j]);
    }//for_i
}//main
```

Ví dụ: Kết quả

```
So phan tu [< 100]: 5
Hay nhap day so...
A[1] = 5
A[2] = 4
A[3] = 3
A[4] = 2
A[5] = 1

Day vua nhap...
   5   4   3   2   1

Sap xep day theo thuat toan lua chon
Luot 1 :    5   4   3   2   1
Luot 2 :    5   4   3   2   1
Luot 3 :    5   4   3   2   1
Luot 4 :    5   4   3   2   1
```

Bài toán chèn phần tử a vào vị trí k

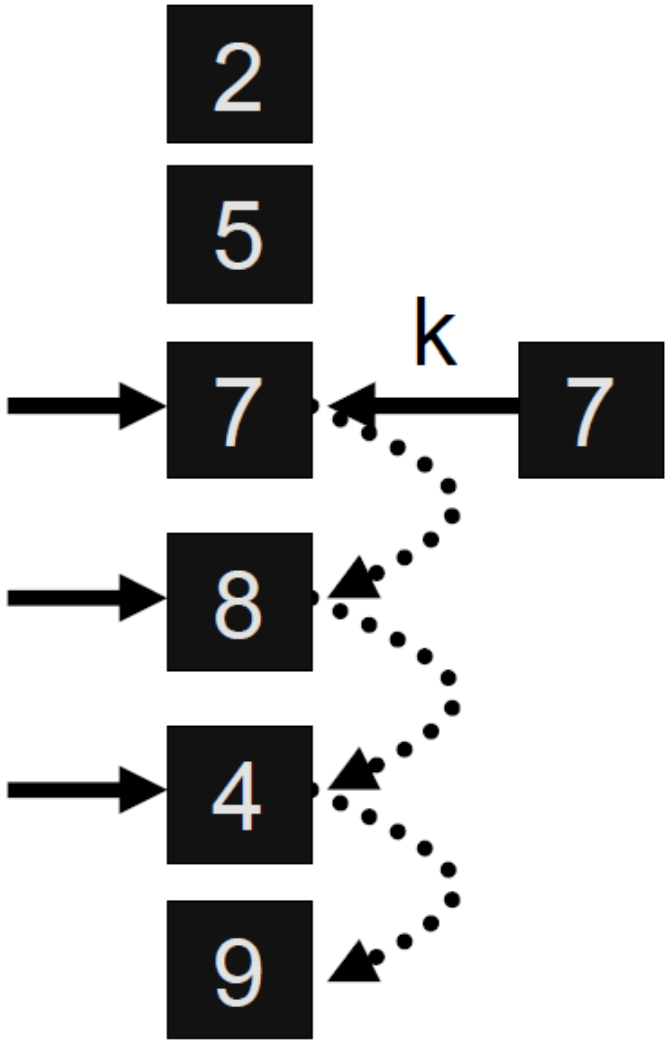
```
for(i = N; i > k; i--)
    A[i] = A[i-1];
A[k] = a;
N= N + 1;
```

Chú ý:

`N = MAX:` không chèn được

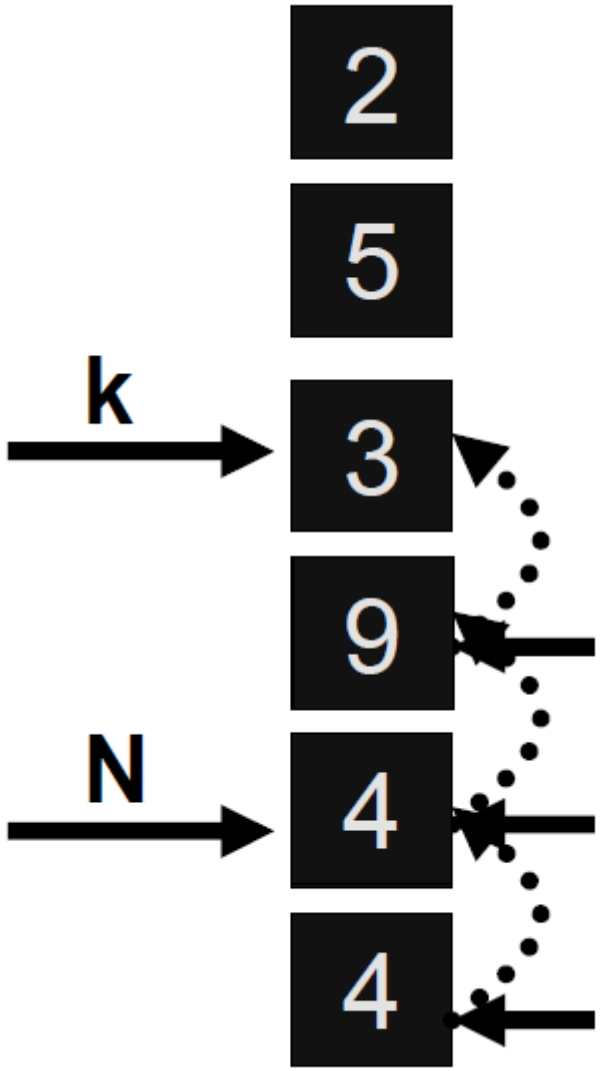
`k > N` → Chèn vào vị trí `N;`

`k< 0` → Chèn vào vị trí 0



Bài toán xóa phần tử ở vị trí k (0≤k<N)

```
for(i = k+1; i < N; i++)
    A[i-1] = A[i];
N= N - 1;
```



Bài tập 1

1) Nhập vào dãy số, tính và đưa ra màn hình
   - Tổng và tích của dãy số
   - Các số chia hết cho 3 và lớn hơn 10
   - Đếm các số nằm trong đoạn [100,1000]
2) Nhập vào một dãy số; tìm số chẵn nhỏ nhất dãy
3) Nhập dãy số; đếm xem có bao nhiêu bộ 3 số thỏa mãn điều kiện $x_i=(x_i-1+x_i+1)/2$
4) Đọc vào dãy số có n phần từ (n<100). Đọc số x và số k nguyên. Chèn x vào vị trí k của dãy. Nếu k>n, chèn x vào vị trí n+1.
5) Nhập vào n và dãy số $(x_1,x_2,\ldots x_n)$;$(y_1,y_2,..y_n)$ rồi tính

$$a)\ \sum_{i=1}^{n} \cos x_i \sin x_i \qquad b)\ \sqrt{\sum_{i=1}^{n}(x_i-y_i)^2} \qquad c)\ \sum_{1}^{n-1} x_2^{i+1} y_{i+1}$$

Bài tập 2

1) Nhập vào từ bàn phím một dãy số nguyên (<100 phần tử). Sắp xếp dãy theo nguyên tắc: Bên trên là số chẵn chia hết cho 3. Bên dưới là số lẻ chia hết cho 3. Giữa là các số còn lại. Đưa cả 2 dãy ra màn hình.
2) Viết chương trình nhập vào từ bàn một dãy số (<100 phần tử). Đưa ra số bé nhất và vị trí những số bằng số bé nhất
3) Nhập vào một dãy số (<100 phần tử) và sắp xếp theo thứ tự tăng dần. Nhập thêm vào một số và chèn số mới nhập vào đúng vị trí
4) Nhập vào một dãy (<100 phần tử); xóa đi các phần tử chia hết cho 5 và đưa kết quả ra màn hình

Giải bài tập – Input & Output Template

```c
#include<stdio.h>
void main(){
    int A[100];
    int N, i;

    //Nhập dữ liệu
    printf("So phan tu : "); scanf("%d",&N);
    for(i=0; i < N; i ++){
        printf("A[%d] = ",i);scanf("%d",&A[i]);
    }
    //Các thao tác xử lý mảng: chèn, xóa, sắp xếp,…
    //TO DO ==

    //Đưa Dữ liệu ra
    for(i=0; i < N; i ++)
        printf("%4d",A[i]);
}
```

Các đoạn code xử lý mảng sẽ được thêm vào đây!

Mọi thao tác Input & Ouput đều như sau cho các bài toán

Bài tập 3: Ma trận

1) Viết chương trình nhập vào một ma trận vuông, các phần tử nguyên:
   - Đưa ra ma trận tam giác dưới
   - Đưa ra ma trận tam giác trên
2) 2. Nhập M, N (M, N < 30) và một ma trận MxN.
3) Đưa ma trận ra màn hình
   - Tìm hàng/cột có tổng các phần tử lớn nhất
   - Tìm số lớn nhất/nhỏ nhất và vị trí trong ma trận
   - Đưa ra ma trận S cùng kích thước thỏa mãn

$$s_{i,j} = \begin{cases} 1 & \text{nê'u } u_{i,j} > 0 \\ 0 & \text{nê'u } u_{i,j} = 0 \\ -1 & \text{nê'u } u_{i,j} < 0 \end{cases}$$

Bài giải: Input & Output Template

```c
#include <stdio.h>
void main(){
    int A[20][20], N,i,j;
    //Nhap Ma Tran
    printf("Nhap kich thuoc : "); scanf("%d",&N);
    printf("\n");
    for ( i=0; i < N; i++)
        for(j=0; j < N; j++) {
            printf("Nhap phan tu [%d,%d]:", i+1,j+1);
            scanf("%d", &A[i][j] );
        }
    //Xuat Ma Tran
    printf("\n\n MA TRAN DA NHAP \n\n");
    for ( i=0; i < N; i++ ){
        for(j=0; j < N; j++)
            printf("%4d" ,A[i][j]);
        printf("\n");
    }}
```

Đưa ra ma trân tam giác trên, dưới

```c
#include <stdio.h>
void main(){
    //Nhap Ma Tran
    //Xuat Ma Tran
    //Đưa ra ma trận tam giác trên, dưới
    printf("\n\n MA TRAN TAM GIAC TREN \n\n");
    for ( i=0; i < N; i++ ){
        for(j=0; j < N; j++)
            if(j >= i)
                printf( "%4d" ,A[i][j]);
            else
                printf("%4c",32); //32 là mã ASCII của dấu cách
        printf("\n");
    }
    printf("\n\n MA TRAN TAM GIAC DUOI \n\n");
    for ( i=0; i < N; i++ ){
        for(j=0; j <= i; j++)
            printf( "%4d" ,A[i][j]);
        printf("\n");
    } }
```

Kết quả

```
Nhap phan tu [1,2]:2
Nhap phan tu [1,3]:3
Nhap phan tu [2,1]:4
Nhap phan tu [2,2]:5
Nhap phan tu [2,3]:6
Nhap phan tu [3,1]:7
Nhap phan tu [3,2]:8
Nhap phan tu [3,3]:9


 MA TRAN DA NHAP

   1   2   3
   4   5   6
   7   8   9
```

```
MA TRAN TAM GIAC TREN

  1   2   3
      5   6
          9


MA TRAN TAM GIAC DUOI

  1
  4   5
  7   8   9
```

Q&A