



Chương 0 Giới thiệu

Bài giảng Kỹ thuật lập trình

Bùi Đức Dương – Khoa Công nghệ thông tin



Thông tin về học phần

- ❖ **Tên học phần: Kỹ thuật lập trình (75326)**
- ❖ **Thời gian: 30 giờ LT + 30 giờ TH + 90 tiết TNC**
- ❖ **Đào tạo trình độ: Đại học, cao đẳng**
- ❖ **Giảng dạy cho sinh viên năm thứ 1 ngành CNTT**





Nội dung của học phần

- ❖ **Chương 1. Thuật toán và chương trình**
- ❖ **Chương 2. Kỹ thuật tổ chức chương trình**
- ❖ **Chương 3. Lập trình hàm**
- ❖ **Chương 4. Các cấu trúc dữ liệu cơ bản**
- ❖ **Chương 5. Kiểu dữ liệu cấu trúc**
- ❖ **Chương 6. Xử lý tập tin**
- ❖ **Chương 7. Lập trình đệ quy**

NTU - Kỹ thuật lập trình



Tài liệu tham khảo

T T	Tác giả	Tên tài liệu	Nơi xuất bản	Địa chỉ
1	Bùi Đức Dương	Bài giảng Kỹ thuật lập trình	Trường Đại học Nha Trang (2014)	CBGD
2	Phạm Văn Át	Kỹ thuật lập trình C	NXB Hồng Đức (2009)	Thư viện
3	Đặng Bình Phương	Tin học cơ sở A	NXB Đại học Quốc gia TP.HCM	Thư viện
4	Khoa CNTT – Trường ĐH KHTN – ĐHQG TP.HCM	Bài tập Nhập môn lập trình	Trường Đại học KHTN - TP.HCM	CBGD
5	Brian W. Kernighan Dennis M.Ritchie	C Programming Language	Prentice Hall Software Series	CBGD

NTU - Kỹ thuật lập trình





Đánh giá học phần

TT	Các nội dung đánh giá	Phương pháp	Tỷ trọng (%)
1	Chuyên cần, xây dựng bài	Điểm danh, quan sát	10
2	Tự nghiên cứu: Đọc tài liệu; làm bài tập	Kiểm tra tại các buổi TH	20
3	Kiểm tra	Viết chương trình trên máy	20
4	Thi cuối kỳ	Viết chương trình trên máy	50

NTU - Kỹ thuật lập trình



Chương 1 Thuật toán và chương trình

Bài giảng Kỹ thuật lập trình



Nội dung

- 1. Các khái niệm cơ bản
- 2. Các bước xây dựng chương trình
- 3. Các phương pháp biểu diễn thuật toán
- 4. Ngôn ngữ lập trình
- 5. Bài tập Chương 1

NTU - Kỹ thuật lập trình



1. Các khái niệm cơ bản

❖ Thuật toán

- Là một **qui tắc** với những dữ liệu đã cho, tìm lời giải sau khoảng **thời gian hữu hạn**
- Là **tập hợp** (dãy) **hữu hạn** các **chỉ thị** (hành động) được định nghĩa rõ ràng nhằm **giải quyết** một bài toán **cụ thể** nào đó.

❖ Lập trình máy tính

- Gọi tắt là **lập trình** (programming).
- Nghệ thuật **cài đặt** một hoặc nhiều **thuật toán** trừu tượng có liên quan với nhau bằng một **ngôn ngữ lập trình** để tạo ra một **chương trình máy tính**.

NTU - Kỹ thuật lập trình





1. Các khái niệm cơ bản

❖ Ví dụ

- Thuật toán giải PT bậc nhất: $ax + b = 0$
(a, b là các số thực).

Đầu vào: a, b thuộc \mathbb{R}
Đầu ra: nghiệm phương trình $ax + b = 0$

- Nếu $a = 0$
 - $b = 0$ thì phương trình có nghiệm bất kì.
 - $b \neq 0$ thì phương trình vô nghiệm.
- Nếu $a \neq 0$
 - Phương trình có nghiệm duy nhất $x = -b/a$



1. Các khái niệm cơ bản

❖ Các tính chất của thuật toán:

- **Tính chính xác:** quá trình tính toán hay các thao tác máy tính thực hiện là chính xác.
- **Tính rõ ràng:** các câu lệnh minh bạch được sắp xếp theo thứ tự nhất định.
- **Tính khách quan:** được viết bởi nhiều người trên máy tính nhưng kết quả phải như nhau.
- **Tính phổ dụng:** có thể áp dụng cho một lớp các bài toán có đầu vào tương tự nhau.
- **Tính kết thúc:** hữu hạn các bước tính toán.





2. Các bước xây dựng chương trình



3. Biểu diễn thuật toán

❖ Sử dụng ngôn ngữ tự nhiên

▪ Ví dụ

Đầu vào: a, b thuộc \mathbb{R}

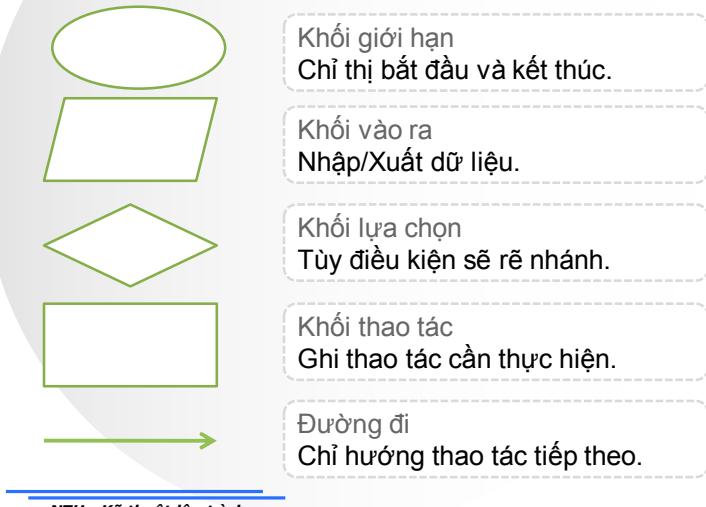
Đầu ra: nghiệm phương trình $ax + b = 0$

1. Nhập 2 số thực a và b .
2. Nếu $a = 0$ thì
 - 2.1. Nếu $b = 0$ thì
 - 2.1.1. Phương trình vô số nghiệm
 - 2.1.2. Kết thúc thuật toán.
 - 2.2. Ngược lại
 - 2.2.1. Phương trình vô nghiệm.
 - 2.2.2. Kết thúc thuật toán.
3. Ngược lại
 - 3.1. Phương trình có nghiệm.
 - 3.2. Giá trị của nghiệm đó là $x = -b/a$
 - 3.3. Kết thúc thuật toán.



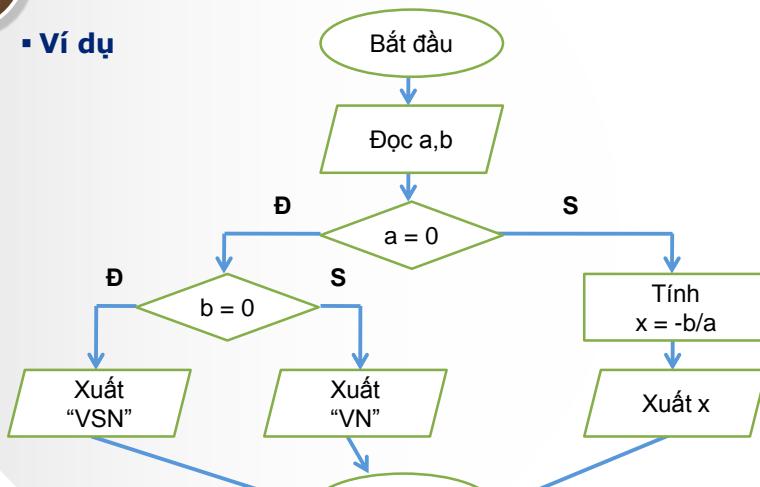
3. Biểu diễn thuật toán

❖ Sử dụng lưu đồ - sơ đồ khối



3. Biểu diễn thuật toán

▪ Ví dụ



NTU - Kỹ thuật lập trình





3. Biểu diễn thuật toán

- ❖ Sử dụng mã giả (Vay mượn ngôn ngữ nào đó để biểu diễn thuật toán)

- Ví dụ

Đầu vào: a, b thuộc \mathbb{R}
Đầu ra: nghiệm phương trình $ax + b = 0$

```
If a = 0 Then
Begin
    If b = 0 Then
        Xuất "Phương trình vô số nghiệm"
    Else
        Xuất "Phương trình vô nghiệm"
End
Else
    Xuất "Phương trình có nghiệm x = -b/a"
```

NTU - Kỹ thuật lập trình



4. Ngôn ngữ lập trình



- ❖ Khái niệm

- **Ngôn ngữ tự nhiên:** giao tiếp người – người
- **Ngôn ngữ lập trình** (programming language):
Hệ thống ký hiệu, quy tắc để “giao tiếp” người – máy
- Ngôn ngữ lập trình được phân ra thành ngôn ngữ lập trình **cấp thấp** và ngôn ngữ lập trình **cấp cao**.
- **Trình biên dịch** (compiler): “dịch” chương trình nguồn thành ngôn ngữ máy.

NTU - Kỹ thuật lập trình





4. Ngôn ngữ lập trình

❖ Ngôn ngữ C/C++

- C là kết quả của quá trình phát triển khởi đầu từ ngôn ngữ BCPL (do Martin Richards đưa ra vào năm 1967)
- Do Dennis Ritchie (tại Bell Telephone) năm 1972.
- Là ngôn ngữ lập trình có cấu trúc
- Hiện nay có gần 30 trình biên dịch C đang phổ biến trên thị trường và chúng không nhất quán nhau. Để cải thiện tình trạng này, chuẩn ANSI C cũng được ra đời vào năm 1978.
- C++(1979), C# được phát triển từ C



4. Ngôn ngữ lập trình

❖ Ưu điểm của C

- **Rất mạnh và linh động**, có khả năng thể hiện bất cứ ý tưởng nào.
- **Được sử dụng rộng rãi** bởi các nhà lập trình chuyên nghiệp.
- **Có tính khả chuyển**, ít thay đổi trên các hệ thống máy tính khác nhau.
- **Rõ ràng, cô đọng**.
- **Lập trình đơn thể**, tái sử dụng thông qua hàm.





4. Ngôn ngữ lập trình

❖ Môi trường phát triển tích hợp IDE (Integrated Development Environment)

- Biên tập chương trình nguồn (Trình EDIT).
- Biên dịch chương trình (Trình COMPILE).
- Chạy chương trình nguồn (Trình RUNTIME).
- Sửa lỗi chương trình nguồn (Trình DEBUG).



❖ Môi trường lập trình

- Dev C++
- Visual Studio 2010

NTU - Kỹ thuật lập trình



5. Bài tập Chương 1

▪ Tự ôn về NNLT C/C++

▪ Đọc TLTK BTCode

▪ Làm bài tập:

1. Viết CT in ra tổng, tích, hiệu và thương của 2 số được nhập vào từ bàn phím.
2. Viết CT in ra trung bình cộng, trung bình nhân của 3 số được nhập vào từ bàn phím.
3. Viết CT nhập cạnh, bán kính và in ra diện tích, chu vi của hình vuông, chữ nhật, tròn.
4. Viết CT nhập cạnh và tính diện tích tam giác.
5. Viết CT nhập số đo 1 góc, cho biết nó thuộc góc vuông thứ mấy trong đường tròn đơn vị.

NTU - Kỹ thuật lập trình





5. Bài tập Chương 1

6. Viết CT giải hệ PTB1
7. Viết CT tính tổng n số nguyên đầu tiên
8. Viết CT kiểm tra n có là số nguyên tố
9. Viết CT tìm USCLN và BSCNN
10. Cho biết kết quả đoạn CT:

```
# include <iostream>
int main()
{
    int a = 5, b = 10, c = 15, d = 0;
    cout << (a+b || c == d);
    cout << (c >= 6) || (a+c <= 20);
    cout << !(b <= 12) && a % 2 == 0;
    cout << !(a > 5) || c < a+b;
    return 0;
}
```

NTU - Kỹ thuật lập trình



Chương 2 Kỹ thuật tổ chức chương trình

Bài giảng Kỹ thuật lập trình



Nội dung

- 1. Tổng quan**
- 2. Các nguyên tắc lập trình**
- 3. Tối ưu hóa mã nguồn**
- 4. Xử lý lỗi và ngoại lệ**
- 5. Bài tập Chương 2**

NTU - Kỹ thuật lập trình



1. Tổng quan

❖ Giới thiệu

- Lập trình được xem như là **"nghệ thuật cài đặt"** các thuật toán bằng một ngôn ngữ lập trình để tạo ra một chương trình máy tính.
- **Kỹ thuật lập trình** trên các ngôn ngữ vì thế tùy thuộc lập trình viên và rất được chú trọng!
- **Hiểu rõ** ngôn ngữ và các nguyên lý lập trình giúp lập trình viên tối ưu chương trình.

```
01011011  
11011110  
00110110  
11001111  
10001111  
10100110  
10001010  
10101011  
00001110  
11010101  
10111010  
01100100  
01010101  
11010110
```



NTU - Kỹ thuật lập trình





1. Tổng quan

❖ Giới thiệu

- Một chương trình nguồn được xem là tốt **không chỉ** được đánh giá thông qua thuật giải đúng và cấu trúc dữ liệu thích hợp **mà còn** phụ thuộc vào phong cách và kỹ thuật mã hoá (coding) của người viết chương trình.
- Nếu một lập trình viên viết một chương trình thực hiện đúng yêu cầu đặt ra nhưng **mã nguồn quá lộn xộn và phong cách lập trình cẩu thả**, thì mã nguồn này sẽ gây khó khăn cho chính người lập trình!
- Vẫn đề phát sinh khi **làm việc với dự án lớn**. Nếu không rèn luyện một phong cách và trang bị một số kỹ thuật lập trình tốt thì LTV đối mặt với nhiều khó khăn...

NTU - Kỹ thuật lập trình



1. Tổng quan

❖ Các giai đoạn của đời sống phần mềm

▪ Đặc tả bài toán

- Phân tích để nắm bắt rõ yêu cầu của bài toán và diễn đạt chính xác lại bài toán bằng ngôn ngữ thích hợp vừa thích ứng với chuyên ngành tin học vừa có tính đại chúng (dễ hiểu đối với nhiều người).

NTU - Kỹ thuật lập trình





1. Tổng quan

❖ Các giai đoạn của đời sống phần mềm

- **Xây dựng hệ thống**

- *Thiết kế:* Xây dựng mô hình hệ thống phần mềm cần có bằng cách phân chia hệ thống thành các module chức năng và xác định rõ chức năng của từng module cũng như mối tương tác giữa các module với nhau. Chức năng của mỗi module được định rõ bởi đặc tả của từng module tương ứng.
- *Triển khai từng module và thử nghiệm :* Viết chương trình cho từng module (bài toán con) thỏa "đúng" đặc tả đã đặt ra.

NTU - Kỹ thuật lập trình



1. Tổng quan

❖ Các giai đoạn của đời sống phần mềm

- **Xây dựng hệ thống**

- *Tính đúng của chương trình được quan tâm bằng 2 hướng khác nhau:*
 - + Chứng minh tính đúng một cách hình thức (thường khó).
 - + Chạy thử chương trình trên nhiều bộ dữ liệu thử khác nhau, mỗi bộ dữ liệu đại diện cho một lớp dữ liệu (thường tồn kém). Để có tính thuyết phục cao, người ta cần chạy thử trên càng nhiều bộ dữ liệu càng tốt. Khi thử nếu phát hiện sai thì phải sửa lại chương trình còn chưa phát hiện sai thì ta con tạm tin chương trình đúng (có tác dụng phát hiện sai và tăng lòng tin vào tính đúng chứ không chứng minh được tính đúng).

NTU - Kỹ thuật lập trình





1. Tổng quan

❖ Các giai đoạn của đời sống phần mềm

- **Xây dựng hệ thống**

- *Thử nghiệm ở mức độ hệ thống:* Sau khi từng module hoạt động tốt, người ta cần thử sự hoạt động phối hợp của nhiều module, thử nghiệm toàn bộ hệ thống phần mềm.



1. Tổng quan

❖ Các giai đoạn của đời sống phần mềm

- **Sử dụng và bảo trì**

- *Sau khi hệ thống phần mềm hoạt động ổn định, người ta đưa nó vào sử dụng.*
- *Trong quá trình sử dụng có thể có những điều chỉnh trong đặc tả của bài toán, hay phát hiện lỗi sai của chương trình. Khi đó cần xem lại chương trình và sửa đổi chúng.*
- *Trong một số dự án, bảo trì chiếm tới hơn 50% tổng số công việc.*





1. Tổng quan

❖ Một số tiêu chuẩn đánh giá chương trình

1. Chương trình đã giải quyết được bài toán đặt ra? (Tính đúng đắn)
2. Chương trình cho kết quả đúng ở mọi trường hợp? (Chính xác)
3. Cùng 1 dữ liệu nhập vào thì kết quả trả ra có luôn luôn giống nhau? (Tin cậy)
4. Có xét tất cả mọi khả năng có thể của dữ liệu nhập vào? (Tổng quát)
5. Chương trình có làm việc tốt ngay cả trong những điều kiện bất thường xảy ra? (An toàn)

NTU - Kỹ thuật lập trình



1. Tổng quan

❖ Một số tiêu chuẩn đánh giá chương trình

6. Chương trình có tiết kiệm tài nguyên, thời gian? (Hiệu quả)
7. Dễ nâng cấp, bảo trì?
8. Giao diện thân thiện? Đầu vào/ra? Nhiều chức năng thay thế? Trợ giúp rõ ràng?
9. Chương trình viết rõ ràng, logic? Cấu trúc dữ liệu hợp lý?
10. Tài liệu hướng dẫn rõ ràng? (tên, đầu vào, đầu ra, giải thích)

NTU - Kỹ thuật lập trình





1. Tổng quan

❖ Các mô hình lập trình

- Lập trình chỉ thị (Imperative Programming)
 - *Pascal, C/C++...*
- Lập trình hướng sự kiện (Event-Driven Programming)
 - *Visual Basic, Java, Asp...*
- Lập trình logic (Logic Programming)
 - *Prolog compiler*
- Lập trình hàm (Functional Programming)
 - *Lisp, ML, Haskell*
- Lập trình tương tranh (Multi-Threading/Parallel)
 - *Data Parallel Programming*

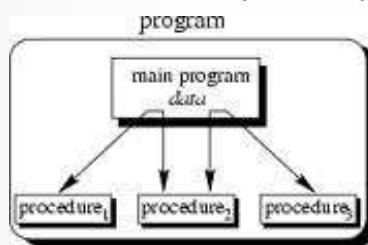
NTU - Kỹ thuật lập trình



1. Tổng quan

❖ Các xu hướng thiết kế chương trình

- Hướng thủ tục (POP: Procedure Oriented Programming)
 - chương trình chia nhỏ thành các chương trình con.
 - sự trừu tượng hóa (Abstraction): chỉ cần biết một hàm đã cho có thể làm được một công việc cụ thể gì?
 - *Chương trình = Cấu trúc dữ liệu + Thuật giải*



NTU - Kỹ thuật lập trình

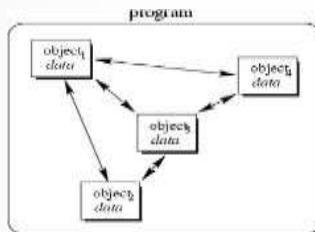




1. Tổng quan

❖ Các xu hướng thiết kế chương trình

- Hướng đối tượng (OOP: Object Oriented Programming)
 - lập trình lấy đối tượng làm nền tảng để xây dựng thuật giải.
 - Đối tượng: dữ liệu + thao tác, hành vi.
 - ***Đối tượng = Phương thức + Dữ liệu***



NTU - Kỹ thuật lập trình

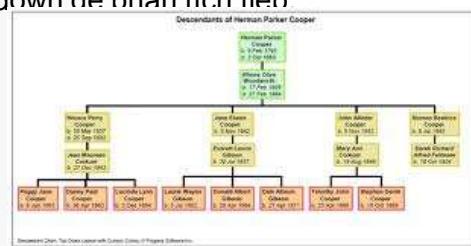


1. Tổng quan

❖ Các xu hướng thiết kế chương trình

➤ Phương pháp thiết kế Top-down

- Quá trình phân tích bài toán được thực hiện **từ trên xuống dưới**.
- Một dự án được **phân tích** thành các chức năng lớn. Sau đó, các chức năng này cũng lại sử dụng nguyên tắc top-down để nhân tích tiến



NTU - Kỹ thuật lập trình





1. Tổng quan

▪ **Ưu điểm:**

- Cấu trúc project được hình dung rõ ràng, liên kết giữa các chức năng được quan tâm hàng đầu.
- Cấu trúc đơn giản, thực hiện dễ dàng

▪ **Nhược điểm:**

- Tuy cấu trúc chương trình đã được phân định rõ ràng, nhưng dữ liệu vẫn dùng chung, chia sẻ. Do vậy, người lập trình cần phải nắm được ý nghĩa, tác dụng của toàn bộ các biến sử dụng trong chương trình hoặc ít nhất là phần có liên quan đến mình. Bất kỳ một sự can thiệp sai vào dữ liệu sẽ gây hậu quả cho mọi phần chương trình khác.



1. Tổng quan

▪ **Ví dụ:**

Chương trình **đọc dữ liệu** từ tập tin thông tin về số nhân viên, thông tin của từng nhân viên của một công ty, **xử lý** và **in ra** thông tin về lương sẽ trả tháng này của từng nhân viên.

➔ Tách chương trình ra thành một số chương trình con đơn giản hơn như sau:

- Xử lý việc đọc tập tin.
- Xử lý thông tin đọc được từ tập tin.
- Xử lý việc thông báo kết quả.

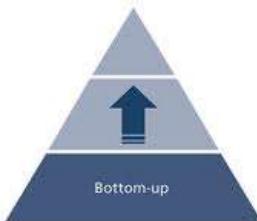




1. Tổng quan

➤ Phương pháp thiết kế Bottom-up

- Quá trình phân tích bài toán được thực hiện từ dưới lên.
- Được sử dụng chủ yếu khi cần xây dựng chương trình từ những thứ đã có sẵn.
- **Ví dụ:** sử dụng các hàm đã có sẵn và được hỗ trợ của ngôn ngữ lập trình để xây dựng một ứng dụng đơn giản nào đó.



NTU - Kỹ thuật lập trình



1. Tổng quan

❖ Thiết kế chương trình trong C/C++

- Khi viết một chương trình trong C/C++, ta có thể triển khai theo **hai cách**:
 - **Cách 1:** Toàn bộ các lệnh của chương trình được viết trong hàm main. Các lệnh được viết theo trình tự để giải quyết bài toán đặt ra.
 - **Cách 2:** Chương trình được tạo thành từ nhiều đơn thể khác nhau. Các đơn thể thực hiện những nhiệm vụ tương đối độc lập và được “lắp ghép” lại thành chương trình thông qua những lời gọi đơn thể trong hàm main.

NTU - Kỹ thuật lập trình





1. Tổng quan

▪ **Ưu nhược điểm:**

- **Cách 1:** Thích hợp khi viết những chương trình có kích thước nhỏ. Toàn bộ thuật toán được thể hiện trong một đoạn mã từ trên xuống dưới. Tuy nhiên, cách này không phù hợp với các chương trình lớn do:
 - *Kích thước chương trình cồng kềnh, khó kiểm soát, chỉnh sửa.*
 - *Các đoạn mã có thể lặp lại, chương trình dài không cần thiết.*
- **Cách 2:** Chương trình được chia nhỏ thành các đơn thể khắc phục được hai nhược điểm cơ bản trên. Đặc biệt phù hợp với các chương trình có kích thước lớn.



1. Tổng quan

❖ **Quy tắc đặt tên:**

- Ngoài tính đúng đắn, chương trình máy tính cần phải dễ đọc và dễ hiểu.
- Hai chuẩn đặt tên đang được sử dụng rộng rãi:
 - Quy tắc đặt tên theo kiểu “lạc đà”
 - Quy tắc đặt tên theo phong cách Hung-ga-ri





1. Tổng quan

▪ Quy tắc đặt tên theo kiểu “lạc đà”

- UpperCamelCase (thường gọi là PascalCase) nếu ký tự đầu tiên của câu được viết hoa

Ví dụ: ThisIsPascalCase

- lowerCamelCase (thường gọi là camelCase) nếu ký tự đầu tiên của câu được viết thường.

Ví dụ: thisIsCamelCase



1. Tổng quan

▪ Quy tắc đặt tên theo phong cách Hung-ga-ri

Tên của biến cho biết kiểu, ý định sử dụng hoặc thậm chí là tầm vực của biến đó:

- Tiền tố viết thường chứa thông tin về kiểu của biến.
- Phần còn lại bắt đầu bằng ký tự hoa cho biết biến chứa thông tin gì.

Ví dụ: iTuSo, fMauSo





2. Các nguyên tắc lập trình

- **Nguyên tắc 1:** *Chương trình nên được tách thành nhiều đơn thể* (mô-đun), mỗi đơn thể thực hiện một công việc và càng độc lập với nhau càng tốt. Điều này sẽ giúp chương trình dễ bảo dưỡng hơn và khi đọc chương trình, ta không phải đọc nhiều, nhớ nhiều các đoạn lệnh nằm rải rác để hiểu được điều gì đang được thực hiện.
- **Nguyên tắc 2:** *Nên sử dụng các tham số* khi truyền thông tin cho các chương trình con. *Tránh sử dụng các biến toàn cục* để truyền thông tin giữa các chương trình con vì như vậy sẽ làm mất tính độc lập giữa các chương trình con và rất khó khăn khi kiểm soát giá trị của chúng khi chương trình thi hành.

NTU - Kỹ thuật lập trình



2. Các nguyên tắc lập trình

- **Nguyên tắc 3:** Cách trình bày chương trình càng *nhất quán* sẽ càng dễ đọc và dễ hiểu. Từ đó, ta sẽ mất ít thời gian để nghĩ về cách viết chương trình và sẽ có nhiều thời gian hơn để nghĩ về các vấn đề cần giải quyết.
- **Nguyên tắc 4:** Chương trình nên thực hiện *như một dòng chảy* từ trên xuống dưới:
 - Các chỉ thị `#include`, `#define` trên cùng.
 - Khai báo các biến toàn cục hay struct.
 - Không nên có những thay đổi bất chợt do sử dụng goto hay continue.

NTU - Kỹ thuật lập trình





2. Các nguyên tắc lập trình

- **Nguyên tắc 5:** Chương trình nên giữ được **tính đơn giản và rõ ràng** trong hầu hết các tình huống. Việc sử dụng các mẹo lập trình chỉ thể hiện sự khéo léo của lập trình viên và làm tăng hiệu quả chương trình lên đôi chút.

- **Ví dụ:**

```

 $\textcircled{Q}$  a[i++] = 1;
 $\textcircled{Q}$  a[i] = 1; i++;

 $\textcircled{Q}$  x += (xp = (2*k < (n - m) ? c[k + 1] : d[k]));
 $\textcircled{Q}$  if (2 * k < n - m)
    xp = c[k + 1];
else
    xp = d[k];
x = x + xp;

```



2. Các nguyên tắc lập trình

- **Nguyên tắc 6:**

- Mỗi dòng lệnh *không nên dài quá 80 ký tự*, giúp việc đọc chương trình dễ dàng hơn khi không phải thực hiện các thao tác cuộn ngang màn hình.
- Mỗi câu lệnh nên được **đặt riêng trên một dòng** để chương trình dễ đọc và dễ quan sát trong tìm lỗi (debug).

- **Ví dụ:**

```

 $\textcircled{Q}$  x=a+b-c*d;for(int i=0; i<n; i++)
 $\textcircled{Q}$  x = a + b - c * d;
for (int i = 0; i < n; i++)

```





2. Các nguyên tắc lập trình

- **Nguyên tắc 7:** Câu lệnh phải thể hiện đúng cấu trúc chương trình bằng cách **sử dụng hợp lý thụt đầu dòng** (tab). (Để cho CT nhìn được rõ ràng và trình bày khoa học, dễ kiểm tra lỗi khi sửa).
- **Ví dụ:**

```

    if (nCount == 0) printf("No data!\n");
    if (nCount == 0)
        printf("No data!\n");
    if (nCount == 0)
        printf("No data!\n");
  
```

NTU - Kỹ thuật lập trình



2. Các nguyên tắc lập trình

- **Nguyên tắc 8:** Các **dấu { }** bao các **khối lệnh** phải được **canh thẳng hàng** theo đặc trưng nhiệm vụ của từng khối lệnh trong bài toán.

Cách 1

```

if (...)

{
...
}

else
{
...
}
  
```

Cách 2

```

if (...) {
...
}

else {
...
}
  
```

- Nên **viết cặp dấu { } trước** rồi viết các lệnh vào giữa để tránh thiếu các dấu ngoặc này.

NTU - Kỹ thuật lập trình





2. Các nguyên tắc lập trình

- **Nguyên tắc 9:** Các câu lệnh nằm giữa cặp dấu {} được viết thụt vào một khoảng tab (các lệnh ngang cấp thì phải thụt vào nhau nhau).

- **Ví dụ:**

```
if (a == 1)
{
    printf("Mot\n");
    if (a == 2)
        printf("Hai\n");
    else
        printf("Khac Mot va Hai!\n");
}
```



NTU - Kỹ thuật lập trình



2. Các nguyên tắc lập trình

- **Nguyên tắc 10:** Các toán tử và toán hạng trong một biểu thức nên được **tách rời nhau bởi 1 khoảng trắng** nhằm làm biểu thức dễ đọc hơn (trừ các toán tử ++, --, -...).

- **Ví dụ:**

```
☺ a=b*c;
☺ a = b * c;

☺ a = b++;
☺ a = b++;

☺ a = - b;
☺ a = -b;
```



NTU - Kỹ thuật lập trình



2. Các nguyên tắc lập trình

- **Nguyên tắc 11:** Nên có khoảng trắng ngăn cách giữa dấu phẩy hay chấm phẩy với các tham số.

- **Ví dụ:** // Khoảng trắng sau dấu , trong khai báo hàm

```

( void hoanVi(int a,int b);
( void hoanVi(int a, int b);
// Khoảng trắng sau dấu , trong Lời gọi hàm
( hoanVi(x,y);
( hoanVi(x, y);
// Khoảng trắng sau dấu ; trong vòng Lặp for
( for (int i = 1;i < n;i++) ...
( for (int i = 1; i < n; i++) ...

```



2. Các nguyên tắc lập trình

- **Nguyên tắc 12:** Nên có khoảng trắng giữa từ khóa và dấu „(“ nhưng không nên có khoảng trắng giữa tên hàm và dấu „,,“

- **Ví dụ:**

```

/ Không nên có khoảng trắng giữa hoanVi và (
( void hoanVi (int a, int b);
( void hoanVi(int a, int b);
//Nên có khoảng trắng giữa if và (
//Không nên có khoảng trắng giữa strcmp và (
( if(strcmp (strInput, "info") == 0)

...
( if (strcmp(strInput, "info") == 0)
...

```





2. Các nguyên tắc lập trình

- **Nguyên tắc 13:** Nên dùng các dòng trống để phân chia các đoạn lệnh trong một hàm như: đoạn nhập/xuất dữ liệu, đoạn tương ứng với các bước xử lý khác nhau.

- **Ví dụ:**

```
void main()
{
    (Các) Đoạn lệnh nhập ở đây
    ...
    (Các) Đoạn lệnh xử lý ở đây
    ...
    (Các) Đoạn lệnh xuất Xuất
    ...
}
```

NTU - Kỹ thuật lập trình



2. Các nguyên tắc lập trình

- **Nguyên tắc 14:** Nên sử dụng các dấu () khi muốn tránh các lỗi về độ ưu tiên toán tử.

- **Ví dụ:**

```
① int i = a >= b && c < d && e <= g + h;
② int i = (a >= b) && (c < d) && (e <= (g + h));
```

NTU - Kỹ thuật lập trình





2. Các nguyên tắc lập trình

- **Nguyên tắc 15:** Tên hằng được viết in và các từ cách nhau bằng dấu _

- **Ví dụ:**

```

    ↪ const int NumberOfElements 100;
    ↪ const int NUMBEROFELEMENTS 100;
    ↪ const int NUMBER_OF_ELEMENTS 100;
  
```

- Các hằng số **không nên viết trực tiếp** (“magic number”) mà nên sử dụng #define hay const để định nghĩa.
- **Không nên dùng #define thường xuyên** để định nghĩa các hằng số, bởi vì trong quá trình debug, ta sẽ không thể xem được giá trị của một hằng số định nghĩa bằng #define.



2. Các nguyên tắc lập trình

- **Nguyên tắc 16:** Tên kiểu tự định nghĩa là danh từ được đặt theo PascalCase và bắt đầu bằng một ký tự đại diện:

- **Ví dụ:**

- TMyTypeName: Kiểu định nghĩa bằng typedef
- SMyStructName: Kiểu cấu trúc (structure)
- UMyUnionName: Kiểu hợp nhất (union)
- EMyEnumName: Kiểu tập hợp (enumeration)

- **Nguyên tắc 17:** Tên biến kiểu dữ liệu tự định nghĩa (typedef, struct, union, class, ...) được viết theo camelCase và không có tiền tố.

- **Ví dụ:**

```

    ↪ CPhanSo psPhanSo;
    ↪ CPhanSo PhanSo;
    ↪ CPhanSo phanSo;
  
```



2. Các nguyên tắc lập trình

- **Nguyên tắc 18:** Tên biến được đặt theo ký hiệu Hung-ga-ri sao cho đủ nghĩa, có thể là là các từ hoàn chỉnh hoặc viết tắt nhưng phải dễ đọc (dễ phát âm)
- **Ví dụ:**

`⌚int ntuso, nmauso;`
`⌚int nTuso, nMauso;`
`⌚ int nTuSo, nMauSo;`
- Biến nên được **khai báo ở gần vị trí** mà nó bắt đầu được sử dụng nhằm tránh việc khai báo các biến dư thừa.
- Trong C chuẩn, tất cả các biến bắt buộc phải khai báo ở đầu khối trước khi sử dụng. Trong C++ biến có thể khai báo ở bất kỳ nơi đâu trước khi sử dụng.

NTU - Kỹ thuật lập trình



2. Các nguyên tắc lập trình

- **Nguyên tắc 19:** Tên của định danh phải **thể hiện được ý nghĩa**: thông thường các biến nguyên như i, j, k dùng làm biến lặp; x, y dùng làm biến lưu tọa độ... Nhìn vào một biến nào đó thì xác định ngay được kiểu dữ liệu mà biến đó lưu trữ. Giả sử có biến đếm số lần thì ta có thể đặt iCount, còn strContent là kiểu chuỗi...
- **Nguyên tắc 20:** Mỗi biến nên khai báo trên một dòng nhằm dễ chú thích về ý nghĩa của mỗi biến.
- Các biến không nên được sử dụng lại với nhiều nghĩa khác nhau trong cùng một hàm.

NTU - Kỹ thuật lập trình





2. Các nguyên tắc lập trình

- **Nguyên tắc 21:** Tên hàm được viết theo camelCase và phải là động từ phản ánh công việc hàm sẽ thực hiện hoặc giá trị trả về của nó.
- **Ví dụ:**

```
int SoLonNhat(int a[], int n);
int TimSoLonNhat(int a[], int n);
int timSoLonNhat(int a[], int n);
```



2. Các nguyên tắc lập trình

- **Nguyên tắc 22:** Nên viết chú thích ngắn gọn nhưng đầy đủ, dễ hiểu. Không nên lạm dụng chú thích.
- Với các chú thích ngắn, đặt trên cùng dòng lệnh; ngược lại đặt câu chú thích trên một dòng riêng phía trên.

- **Ví dụ:**

```
if (a == 2)
    return true; // Trả về giá trị
else
    // Mô tả và xử lý tiếp theo...
(Các thao tác tiếp theo)
```





2. Các nguyên tắc lập trình

- **Nguyên tắc 23:** Nên viết biểu thức điều kiện mang tính tự nhiên, biểu thức nên viết dưới dạng khẳng định
- **Ví dụ:**

```
if ( !(x1 < x2) || !(y1 >= y2))
    if ( (x1 >= x2) || (y1 < y2))
```
- **Nguyên tắc 24:** Dùng chính ngôn ngữ đó để tính kích thước của đối tượng.
 - Khi cần lấy kích thước của biến int, ta có thể dùng sizeof(int) thay cho các giá trị 2 hay 4.
 - Tương tự như vậy khi lấy kích thước của phần tử trong một mảng int ta dùng sizeof(array[0]) thay cho sizeof(int).

NTU - Kỹ thuật lập trình



3. Tối ưu hóa mã nguồn

- Mã nguồn nếu được viết tốt sẽ làm cho **tốc độ chương trình cải thiện đáng kể**.
- Có thể ngày nay năng lực xử lý của máy tính khá mạnh, do đó người lập trình không quan tâm đến việc tối ưu mã nguồn. Nhưng cũng **không vì thế mà bỏ qua kỹ thuật này**.
- Việc **cài đặt cũng cần phải có kỹ thuật**, nếu không thì chính khả năng cài đặt của lập trình viên làm hạn chế sự thực thi của thuật giải hay chương trình.
- **Mục đích của việc tối ưu mã nguồn** là nâng cao tốc độ xử lý và hạn chế không gian bộ nhớ mà chương trình chiếm dụng. Thông thường có thể mâu thuẫn giữa tốc độ và không gian lưu trữ, do đó tuỳ theo điều kiện cụ thể mà người lập trình có thể lựa chọn thích hợp.

NTU - Kỹ thuật lập trình





3. Tối ưu hóa mã nguồn

- **Thu gọn những biểu thức dùng nhiều lần:**

- Nếu một biểu thức tính toán được dùng nhiều lần thì ta nên tính kết quả một lần rồi lưu vào một biến và dùng lại. Hạn chế việc tính toán với cùng một biểu thức nhiều lần!

- **Ví dụ:**

+ $F = \sqrt{dx*dx+dy*dy} + (\sqrt{dx*dx + dy*dy}*\sqrt{dx*dx}-\sqrt{dy*dy})...$

+ Trong dãy biểu thức trên có $\sqrt{dx*dx+dy*dy}$, $dx*dx$, $dy*dy$ được dùng nhiều chỗ, ta có thể tính trước bên ngoài và lưu vào biến tạm để dùng lại sau này.

- **Ví dụ:** Delta trong PTB2



3. Tối ưu hóa mã nguồn

- **Đưa những biểu thức không phụ thuộc vòng lặp ra ngoài:**

- Trong một số vòng lặp ta có sử dụng biểu thức tính toán nhưng giá trị của biểu thức không phụ thuộc vào sự thay đổi của vòng lặp thì có thể đưa biểu thức này ra ngoài.

- **Ví dụ:**

```
for(i = 0; i < strlen(str); i++)
...
...
```

chuyển thành:

```
int n = strlen(str);
for(i = 0; i < n; i++)
...
...
```





3. Tối ưu hóa mã nguồn

- **Thay thế một biểu thức bằng một biểu thức tương đương nhưng lợi về thực thi:**

- **Ví dụ:** trong chương trình xử lý ảnh đòi hỏi tốc độ cao, ta có thể thay thế các phép nhân chia bằng phép dịch chuyển bit; Thay thế sử dụng chỉ mục trong mảng bằng con trỏ...
- **Ví dụ:** khi so sánh khoảng cách của hai điểm như sau:

```
if(sqrt(dx1*dx1+dy1*dy1) < sqrt(dx2*dx2+dy2*dy2))
...
...
```

chuyển thành:

```
if ((dx1*dx1+dy1*dy1) < (dx2*dx2+dy2*dy2))
...
...
```



3. Tối ưu hóa mã nguồn

- **Dùng số nguyên thay cho số thực:**

- Do việc xử lý số thực chậm hơn xử lý số nguyên nên ta có thể dùng số nguyên thay cho số thực có phần lẻ nhỏ.
- **Ví dụ:** điểm trung bình của sinh viên là số thực ta có thể thay bằng số nguyên:
+ DTB là 8.72 thì lưu số nguyên 872, khi xuất ra thì chia cho 100.

- **Tránh lãng phí bộ nhớ:** bằng cách sử dụng kiểu dữ liệu nhỏ nhất có thể được để lưu trữ. Tốc độ chương trình sẽ nhanh hơn khi sử dụng kiểu dữ liệu nhỏ hơn.
- **Khai báo biến cục bộ trong phạm vi gần nhất:** Việc khai báo ở phạm vi rộng hơn chỉ làm lãng phí và khó kiểm soát.





3. Tối ưu hóa mã nguồn

- **Loại bỏ vòng lặp:**

- Nếu thân vòng lặp đơn giản và số lần lặp cũng không nhiều, ta có thể làm cho đoạn chương trình hiệu quả hơn bằng cách bỏ vòng lặp.

- **Ví dụ:**

```
for(i = 0; i < 3; i++)
    A[i] = B[i] + C[i];
```

chuyển thành:

```
A[1] = B[1] + C[1];
A[2] = B[2] + C[2];
A[3] = B[3] + C[3];
```

NTU - Kỹ thuật lập trình



3. Tối ưu hóa mã nguồn

- **Loại rẽ nhánh trong vòng lặp**

```
for(i=0;i<M;i++)
{
    a[i]=a[i]+b[i];
    if(f) a[i]=0;
}
```



```
if(f)
for(i=0;i<M;i++)
{
    a[i]=a[i]+b[i];
    a[i]=0;
}
else
for(i=0;i<M;i++)
    a[i]=a[i]+b[i];
```

NTU - Kỹ thuật lập trình





3. Tối ưu hóa mã nguồn

- Thoát khỏi vòng lặp sớm nhất

```
found = FALSE;
for(i=0;i<M;i++)
    if(a[i]==X)
        found=TRUE;
if(found)
    printf("There is a X");
```



```
found = FALSE;
for(i=0;i<M;i++)
    if(a[i]==X)
    {
        found=TRUE;
        break;
    }
if(found)
    printf("There is a f");
```



3. Tối ưu hóa mã nguồn

- Gom các vòng lặp cùng số lần lặp

- Ví dụ:

```
for (i=1; i<n; i++) s = s + a[i];
for (i=1; i<n; i++) t = t * a[i];
```

chuyển thành một vòng lặp với 2 biểu thức bên trong

- Sử dụng phép shift thay cho nhân và chia

- Shift trái 1 bit: nhân 2, Shift phải 1 bit: chia 2

- Ví dụ:

```
a *= 4 ⇒ a<<2
a = 8*(b+c) ⇒ a = (b+c)<<3
```



3. Tối ưu hóa mã nguồn

- **Sử dụng macro:** một số hàm đơn giản và thường sử dụng có thể chuyển thành macro để tăng tốc độ thực thi. Do mỗi lần gọi hàm sẽ tốn chi phí cho việc gọi và trả về từ hàm.

- **Ví dụ:**

```
int max(int a, int b)
{
    return a>b? a: b;
}
```

```
#define max(a, b) ((a)>(b)) ? (a) : (b)
```



3. Tối ưu hóa mã nguồn

- **Giảm số lượng tham số truyền vào hàm:** việc sử dụng hàm có quá nhiều tham số được truyền vào có thể làm ảnh hưởng đến ngăn xếp dành cho việc gọi hàm. Nhất là trường hợp tham số là kiểu dữ liệu cấu trúc. Sử dụng con trỏ hay tham chiếu trong trường hợp này để đơn giản hoá.

- **Ví dụ:**

```
void Print(struct Student s)
{ printf("%d", s.StudentCode);... }
```

```
void Print(const struct Student *s)
{ printf("%d", s->StudentCode);... }
```



4. Xử lý lỗi và ngoại lệ

❖ Lỗi và ngoại lệ

- Các lỗi thường gặp
 - Lỗi từ vựng (Token).
 - Lỗi cú pháp (Syntax).
 - Lỗi ngữ nghĩa (Semantic).
- Mọi đoạn chương trình đều tiềm ẩn khả năng sinh lỗi.
 - Lỗi chủ quan: do lập trình sai.
 - Lỗi khách quan: do dữ liệu, do trạng thái của hệ thống.
- Ngoại lệ: các trường hợp hoạt động không bình thường.

NTU - Kỹ thuật lập trình



4. Xử lý lỗi và ngoại lệ

❖ Hạn chế lỗi

- Sử dụng hàm khi lập trình
- Dùng Comment (Chú thích)
- Đặt tên các variables có ý nghĩa
- Soạn một Test Plan
- Xử lý Error lúc Run time
- Dùng Breakpoints
- Dùng Watch Window

NTU - Kỹ thuật lập trình





4. Xử lý lỗi và ngoại lệ

❖ Xử lý ngoại lệ

- Dựa trên cơ chế ném và bắt ngoại lệ.
 - Ném ngoại lệ: dừng chương trình và chuyển điều khiển lên mức trên (nơi bắt ngoại lệ).
 - Bắt ngoại lệ: xử lý với ngoại lệ.
- Ngoại lệ: là đối tượng mang thông tin về lỗi đã xảy ra.
 - Ngoại lệ được ném tự động.
 - Ngoại lệ được ném tường minh.

NTU - Kỹ thuật lập trình



5. Bài tập Chương 2

1. Tìm hiểu về các mô hình lập trình:

- a. Lập trình chỉ thị
- b. Lập trình hướng sự kiện
- c. Lập trình logic
- d. Lập trình hàm
- e. Lập trình tương tranh

Chọn 1 trong 4 PP từ b-e để cài đặt 1 bài toán

2. Tìm hiểu về lập trình hướng đối tượng

3. Lấy ví dụ cho mỗi nguyên tắc lập trình

4. Lấy ví dụ cho mỗi PP tối ưu mã nguồn

5. Tìm hiểu về các phương pháp kiểm thử

NTU - Kỹ thuật lập trình





Chương 3

Lập trình hàm

Bài giảng Kỹ thuật lập trình





1. Lập trình cấu trúc

- ❖ Lập trình cấu trúc là còn được hiểu là **lập trình thủ tục**.
- ❖ Theo cách tiếp cận hướng thủ tục thì hệ thống phần mềm được xem như là **dãy các công việc** cần thực hiện như đọc dữ liệu, tính toán, xử lý, lập báo cáo và in ấn kết quả... Mỗi công việc đó sẽ được thực hiện bởi một hàm hay thủ tục nhất định.
- ❖ Phương pháp lập trình cấu trúc thường đi đôi với **phương pháp phân tích trên xuống** (top-down).
- ❖ Các **ngôn ngữ** hỗ trợ lập trình hướng cấu trúc phổ biến là Pascal, C, Foxpro.

NTU - Kỹ thuật lập trình



1. Lập trình cấu trúc

- ❖ Phương pháp thủ tục **chia** một chương trình (chức năng) lớn thành các khối chức năng hay hàm (thủ tục) đủ nhỏ để dễ lập trình và kiểm tra.
- ❖ Mỗi hàm có **một điểm bắt đầu và một điểm kết thúc** và có dữ liệu và logic riêng, làm việc độc lập. Trong chương trình, các biến có các phạm vi hoạt động nhất định.
- ❖ Dữ liệu được chuyển đổi qua lại **qua các tham số** gọi hàm. Việc chia chương trình thành các hàm cho phép nhiều người có thể tham gia vào việc xây dựng chương trình.

NTU - Kỹ thuật lập trình





1. Lập trình cấu trúc

- ❖ Phương pháp này dẫn đến một khái niệm mới – **sự trừu tượng hóa**. Sự trừu tượng hóa có thể xem như khả năng quan sát một sự việc mà không cần xem xét đến các chi tiết bên trong của nó.
- ❖ Một chương trình có thể xây dựng dựa trên 3 cấu trúc:
 - **Trình tự**: các câu lệnh được thực hiện theo trình tự nhất định (trên xuống).
 - **Quyết định (rẽ nhánh)**: qui định sẽ thực hiện chương trình như thế nào phụ thuộc vào sự thỏa mãn các điều kiện nhất định.
 - **Vòng lặp**: sự thực hiện có tính lặp một số đoạn lệnh của chương trình khi các điều kiện nào đó vẫn được thỏa mãn.

NTU - Kỹ thuật lập trình



1. Lập trình cấu trúc

- ❖ Lập trình hướng cấu trúc rất phổ biến trong những năm 1980 và đầu những năm 1990.
- ❖ Do những hạn chế khi lập trình hệ thống lớn, lập trình hướng cấu trúc đã dần bị thay thế bởi lập trình hướng đối tượng.
- ❖ Cho đến nay, ngôn ngữ lập trình hướng cấu trúc thường được sử dụng để dạy học và cho những chương trình nhỏ.

NTU - Kỹ thuật lập trình





2. Xây dựng hàm

❖ Khái niệm về hàm

- Một đoạn chương trình có tên, đầu vào và đầu ra.
- Có chức năng giải quyết một số vấn đề chuyên biệt cho chương trình chính.
- Được gọi nhiều lần với các tham số khác nhau.
- Được sử dụng khi có nhu cầu:
 - Tái sử dụng.
 - Sửa lỗi và cải tiến.

NTU - Kỹ thuật lập trình



2. Xây dựng hàm

❖ Cú pháp hàm

```
<kiểu trả về> <tên hàm>([danh sách tham số])
{
    <các câu lệnh>
    [return <giá trị>]
}
```

- Trong đó
 - <kiểu trả về> : kiểu bất kỳ của C (**char**, **int**, **long**, **float**,...). Nếu không trả về thì là **void**.
 - <tên hàm>: theo quy tắc đặt tên định danh.
 - <danh sách tham số> : **tham số hình thức đầu vào** giống khai báo biến, cách nhau bằng dấu ,
 - <giá trị> : trả về cho hàm qua lệnh **return**.

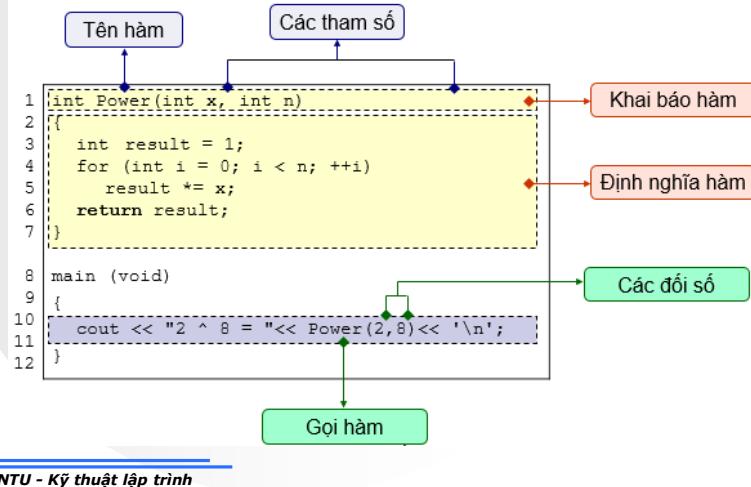
NTU - Kỹ thuật lập trình





2. Xây dựng hàm

❖ Ví dụ



2. Xây dựng hàm

❖ Các thông tin cho việc xây dựng hàm:

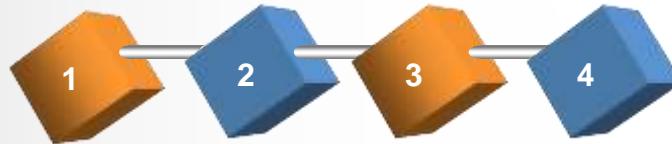
- Tên hàm.
- Hàm sẽ thực hiện công việc gì.
- Các đầu vào (nếu có).
- Đầu ra (nếu có).





2. Xây dựng hàm

❖ Các bước xây dựng chương trình sử dụng hàm:



Phân chia
chương
trình
thành các
vấn đề
nhỏ.

Với mỗi vấn
đề, xác định
các thành
phần: Tên
hàm, kết
quả trả về,
tham số

Chuyển
từng vấn
đề thành
các chương
trình con
(hàm)

Gọi các
hàm theo
trình tự
hợp lý

NTU - Kỹ thuật lập trình



2. Xây dựng hàm

❖ Ví dụ

- **Tên hàm:** TinhTong
- **Công việc:** tính và trả về tổng 2 số nguyên
- **Đầu vào:** hai số nguyên x và y
- **Đầu ra:** một số nguyên có giá trị $x + y$

```
int TinhTong(int x, int y)
{
    int s;
    s = x + y;
    return s;
}
```

NTU - Kỹ thuật lập trình





2. Xây dựng hàm

- ❖ Thông thường người ta thường đặt phần tiêu đề hàm/nguyên mẫu hàm (prototype) trên hàm main và phần định nghĩa hàm dưới hàm main.

```
void XuatTong(int x, int y); // prototype

void main()
{
    ...
}

void XuatTong(int x, int y)
{
    printf("%d cong %d bang %d", x, y, x + y);
}
```

NTU - Kỹ thuật lập trình



2. Xây dựng hàm

❖ Tầm vực

- Là phạm vi hiệu quả của biến và hàm.
- Biến:
 - **Toàn cục:** khai báo trong ngoài tất cả các hàm (kể cả hàm main) và có tác dụng lên toàn bộ chương trình.
 - **Cực bộ:** khai báo trong hàm hoặc khối {} và chỉ có tác dụng trong bản thân hàm hoặc khối đó (kể cả khối con nó). Biến cục bộ sẽ bị xóa khỏi bộ nhớ khi kết thúc khối khai báo nó.

NTU - Kỹ thuật lập trình





3. Truyền tham số

❖ Truyền Giá trị (Call by Value)

- Truyền đối số cho hàm ở dạng giá trị.
- Có thể truyền hằng, biến, biểu thức nhưng **hàm chỉ sẽ nhận giá trị**.
- Được sử dụng khi **không có nhu cầu thay đổi giá trị của tham số** sau khi thực hiện hàm.

```
void TruyenGiaTri(int x)
{
    ...
    x++;
}
```

NTU - Kỹ thuật lập trình



3. Truyền tham số

❖ Truyền Địa chỉ (Call by Address)

- Truyền đối số cho hàm ở dạng địa chỉ (con trỏ).
- **Không được truyền giá trị** cho tham số này.
- Được sử dụng khi có **nhu cầu thay đổi giá trị** của tham số sau khi thực hiện hàm.

```
void TruyenDiaChi(int *x)
{
    ...
    *x++;
}
```

NTU - Kỹ thuật lập trình





3. Truyền tham số

❖ Truyền Tham chiếu (Call by Reference) (C++)

- Truyền đối số cho hàm **ở dạng địa chỉ** (con trỏ). Được bắt đầu bằng & trong khai báo.
- **Không được truyền giá trị** cho tham số này.
- Được sử dụng khi **có nhu cầu thay đổi giá trị của tham số** sau khi thực hiện hàm.

```
void TruyenThamChieu(int &x)
{
    ...
    x++;
}
```

NTU - Kỹ thuật lập trình



3. Truyền tham số

❖ Lưu ý

- Sử dụng **tham chiếu** là một cách để trả về giá trị cho **chương trình**.

```
int TinhTong(int x, int y)
{
    return x + y;
}
void TinhTong(int x, int y, int &tong)
{
    tong = x + y;
}
void TinhTongHieu(int x, int y, int &tong, int &hieu)
{
    tong = x + y; hieu = x - y;
}
```

NTU - Kỹ thuật lập trình





3. Truyền tham số

❖ Cách gọi hàm

- Gọi tên của hàm đồng thời truyền các đối số (hằng, biến, biểu thức) cho các tham số theo đúng thứ tự đã được khai báo trong hàm.
- Các biến hoặc trị này cách nhau bằng dấu ,
- Các đối số này được đặt trong cặp dấu ngoặc đơn ()
- <đối số 1>, ..., <đối số n>;



3. Truyền tham số

❖ Ví dụ

```
void HoanVi(int &a, int &b);

void main()
{
    HoanVi(2912, 1706);
    int x = 2912, y = 1706;
    HoanVi(x, y);
}

void HoanVi(int &a, int &b)
{
    int tam = a;
    a = b;
    b = tam;
}
```





4. Một số dạng hàm đặc biệt

❖ Các đối số của chương trình

- Hàm **main** là hàm nên **cũng có tham số**.
- Chương trình tự động thực hiện hàm main mà không cần lời gọi hàm.
→ **Làm sao truyền đối số?**
→ Khi thực thi tập tin chương trình (.exe), ta truyền kèm đối số. Tất nhiên, hàm **main** cũng phải định nghĩa các tham số để có thể nhận các đối số này.



4. Một số dạng hàm đặc biệt

❖ Các tham số của hàm main

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

- Trong đó
 - **argc** là số lượng đối số (**tính luôn tên tập tin chương trình**)
 - **argv** là mảng chứa các đối số (**dạng chuỗi**)





4. Một số dạng hàm đặc biệt

❖ Ví dụ

- Viết chương trình có tên **Cong**, nhận 2 đối số **x** và **y** và xuất ra giá trị **x + y**.

```
argv = {"Cong.EXE", "2912", "1706";
```

```
argc = 3
```



4. Một số dạng hàm đặc biệt

❖ Ví dụ

- Viết chương trình có tên **Cong**, nhận 2 đối số **x** và **y** và xuất ra giá trị **x + y**.

```
#include <stdio.h>
#include <stdlib.h>      // atoi
void main(int argc, char *argv[]) {
    if (argc == 3) {
        int x = atoi(argv[1]);
        int y = atoi(argv[2]);
        printf("%d + %d = %d", x, y, x+y);
    }
    else
        printf("Sai! VD: Cong 2912 1706");
}
```





4. Một số dạng hàm đặc biệt

❖ Ví dụ

- Viết chương trình có tên **test** nhận dữ liệu từ tập tin **input.txt**, xử lý và xuất kết quả ra tập tin **output.txt**.

```
argv = {"test", "input.txt", "output.txt"};
```

```
argc = 3
```



4. Một số dạng hàm đặc biệt

❖ Ví dụ

- Viết chương trình có tên **test** nhận dữ liệu từ tập tin **input.txt**, xử lý và xuất kết quả ra tập tin **output.txt**.

```
#include <stdio.h>
void main(int argc, char *argv[]) {
    if (argc == 3) {
        // Nhập dữ liệu từ tập tin argv[1]
        // Xử lý
        // Xuất kết quả ra tập tin argv[2]
    }
    else
        printf("Sai! VD: test in.txt out.txt");
}
```





4. Một số dạng hàm đặc biệt

❖ Ví dụ

- Viết hàm **Tong** để tính tổng 4 số x, y, z, t

```
int Tong(int x, int y, int z, int t)
{
    return x + y + z + t;
}
```

- Tính tổng 4 số 2912, 1706, 1506, 1904

Tong(2912, 1706, 1506, 1904);

- Nếu chỉ muốn tính tổng 2 số 2912, 1706

Tong(2912, 1706, 0, 0); // z = 0, t = 0

NTU - Kỹ thuật lập trình



4. Một số dạng hàm đặc biệt

❖ Hàm có đối số mặc định

- Hàm có đối số mặc định là hàm có một hay nhiều tham số hình thức được gán giá trị.
- Tham số này nhận giá trị mặc định đó nếu không có đối số truyền vào cho tham số đó.
- Phải được dồn về **tận cùng bên phải**.

❖ Ví dụ

```
int Tong(int x, int y, int z = 0, int t = 0)
{
    return x + y + z + t;
}
```

NTU - Kỹ thuật lập trình





4. Một số dạng hàm đặc biệt

❖ Lưu ý

- Muốn truyền đối số khác thay cho đối số mặc định, phải truyền đối số thay cho các đối số mặc định trước nó.

```
int Tong(int x, int y = 0, int z = 0);
```

```
int Tong(1, 5);
```

```
int Tong(1, 0, 5);
```



4. Một số dạng hàm đặc biệt

❖ Ví dụ

- In thông tin SV trong lớp gồm: họ tên, phái, lớp, năm sinh

```
void XuatThongTin(char *hoten, char phai = 0,
char *lop = "TH07", int namsinh = 1989)
{
    puts(hoten);
    printf(phai == 0? "Nam\n" : "Nu\n");
    puts(lop);
    printf("%d", namsinh);
}
```





4. Một số dạng hàm đặc biệt

❖ Ví dụ

- In thông tin SV trong lớp gồm: họ tên, phái, lớp, năm sinh

```
void main()
{
    XuatThongTin("Nguyen Van A");
    XuatThongTin("Tran Thi B", 1);
    XuatThongTin("Hoang Van C", 0, "TH00");
    XuatThongTin("Le D", 1, "TH07", 1988);
}
```



4. Một số dạng hàm đặc biệt

❖ Nhận xét

- **x = a** thường xuyên xảy ra thì nên chuyển x thành tham số có đối số mặc định là a.
Ví dụ, hầu hết **phai = 0** (nam), **lop = "TH07"** và **namsinh = 1989**.
- **x = a** và **y = b** thường xuyên xảy ra nhưng **y = b** thường xuyên hơn thì nên đặt tham số mặc định x trước y.
Ví dụ, **lop = "TH07"** xảy ra nhiều hơn **phai = 0** nên đặt **lop sau phai**.





4. Một số dạng hàm đặc biệt

❖ Chỉ thị tiền xử lý #define

- Chỉ thị **#define <name> <value>**
- Mọi chỗ xuất hiện **<name>** trong chương trình nguồn được thay thế bằng **<value>** để tạo ra chương trình tiền xử lý.
- Ví dụ
 - `#define MAX 1000`
 - `#define PI 3.14`
 - `#define message "Hello World\n"`



4. Một số dạng hàm đặc biệt

❖ Định nghĩa các macro (lệnh gộp - lệnh tắt)

- **#define <name>(<param-list>) <expression>**
- Mọi chỗ xuất hiện của **<name>** với lượng tham số đưa vào phù hợp sẽ được thay thế bởi **<expression>** (tham số được thay thế tương ứng)
- Ví dụ
 - `#define showmsg(msg) printf(msg)`
 - `→showmsg("Hello"); ⇔ printf("Hello");`





4. Một số dạng hàm đặc biệt

❖ Hàm nội tuyến (inline)

- Xét 2 cách sau

```
#define PI 3.14159
float addPi(float s)
{
    return s + PI;
}

void main()
{
    float s = 0;
    for (int i = 1; i<=100000; i++)
        s = s + PI;           // Cách 1 (0.7s)
        s = addPi(s);         // Cách 2 (1.4s)
}
```

NTU - Kỹ thuật lập trình



4. Một số dạng hàm đặc biệt

❖ Nhận xét

- Sử dụng hàm giúp chương trình dễ hiểu nhưng lại tốn chi phí cho lời gọi hàm.

❖ Khắc phục

- Sử dụng hàm nội tuyến (inline) bằng cách thêm từ khóa inline trước prototype của hàm.

➔ **inline** float addPi(float s) {return s + PI;}

❖ Khái niệm

- Sao chép thân hàm đến bất cứ nút nào hàm được gọi ➔ **kết quả giống hệt cách 1.**

NTU - Kỹ thuật lập trình





4. Một số dạng hàm đặc biệt

❖ Lưu ý

- **Giảm thời gian** thực hiện hàm (gọi và kết thúc).
- **Giảm không gian** bộ nhớ do các hàm con chiếm dụng khi hàm được gọi.
- **Không** cho phép các hàm nội tuyến đệ quy.
- **Phản lớn không** cho phép thực hiện nội tuyến các hàm sử dụng vòng lặp while.
- **Chỉ inline các hàm nhỏ**, inline các hàm lớn sẽ gây phản tác dụng (bộ nhớ cho hàm inline chiếm giữ sẽ lâu giải phóng hơn).



4. Một số dạng hàm đặc biệt

❖ Hàm trả về tham chiếu

- Hàm chỉ trả về giá trị. Ví dụ, `x = f();`
- Vậy, `g() = x` hợp lệ hay không?
- → Hợp lệ khi `g(x)` trả về tham chiếu đến một biến (C++)

❖ Cú pháp

```
<kiểu trả về> &<tên hàm>([<ds tham số>])
{
    return <biến>;
}
```





4. Một số dạng hàm đặc biệt

❖ Ví dụ

```
#include <stdio.h>

int x;

int &getx()
{
    return x;
}

void main()
{
    getx() = 5; // ⇔ x = 5
}
```

NTU - Kỹ thuật lập trình



4. Một số dạng hàm đặc biệt

❖ Ứng dụng

- Chỉ số của mảng trong C/C++ bắt đầu từ 0
➔ Không quen thuộc lắm.
- Viết hàm để khi muốn truy cập đến phần tử thứ i của mảng a ta sử dụng V(i) thay vì a[i-1]

```
int a[100];
int &V(int i)
{
    return a[i-1];
}
...
V(1) = 2912; // ⇔ a[0] = 2912;
```

NTU - Kỹ thuật lập trình





4. Một số dạng hàm đặc biệt

❖ Chú ý

- Trong trường hợp sau, biến x phải là biến toàn cục → không nên sử dụng!

```
int x; // biến toàn cục

int &getx()
{
    return x;
}

void main()
{
    getx() = 2912;
}
```

NTU - Kỹ thuật lập trình



4. Một số dạng hàm đặc biệt

❖ Chú ý

- Nếu không muốn sử dụng biến toàn cục, phải **truyền x ở dạng tham chiếu**.

```
int &getx(int x) { // SAI! x là tham trị → bản sao
    return x;
}

int &getx() {
    int x;           // SAI! x là biến cục bộ
    return x;
}

int &getx(int &x) { // ĐÚNG! x là tham chiếu
    return x;
}
```

NTU - Kỹ thuật lập trình





4. Một số dạng hàm đặc biệt

❖ Ví dụ

```
#include <stdio.h>

int &V(int a[], int i)
{
    return a[i-1];
}

void main()
{
    int a[100];
    for (int i = 1; i <= 100; i++)
        V(a, i) = 0;
}
```

NTU - Kỹ thuật lập trình



4. Một số dạng hàm đặc biệt

❖ Khuôn mẫu hàm

- Viết hàm tìm số nhỏ nhất trong 2 số
 - Viết các hàm khác nhau để tìm min 2 số int, 2 số long, 2 số float, 2 số double, 2 phân số...
- Nhược điểm
 - Hàm bản chất giống nhau nhưng khác kiểu dữ liệu nên phải viết nhiều hàm giống nhau.
 - Sửa 1 hàm phải sửa những hàm còn lại.
 - Không thể viết đủ các hàm cho mọi trường hợp do còn nhiều kiểu dữ liệu khác.

NTU - Kỹ thuật lập trình





4. Một số dạng hàm đặc biệt

❖ Khái niệm

- Viết một hàm duy nhất nhưng có thể sử dụng cho nhiều kiểu dữ liệu khác nhau.

❖ Cú pháp

- **template <ds mẫu tham số> <khai báo hàm>**

❖ Ví dụ

```
template <class T> <khai báo hàm>  
hoặc  
template <class T1, class T2> <khai báo hàm>
```

NTU - Kỹ thuật lập trình



4. Một số dạng hàm đặc biệt

❖ Ví dụ

```
template <class T>  
T min(T a, T b)  
{  
    if (a < b)  
        return a;  
    return b;  
}  
  
void main()  
{  
    int a = 2912, b = 1706;  
    int m = min<int>(a, b);  
    printf("So nho nhat la %d", m);  
}
```

NTU - Kỹ thuật lập trình





4. Một số dạng hàm đặc biệt

❖ Lợi ích của việc sử dụng khuôn mẫu hàm

- Dễ viết, do chỉ cần viết hàm tổng quát nhất.
- Dễ hiểu, do chỉ quan tâm đến kiểu tổng quát nhất.
- Có kiểu an toàn do trình biên dịch kiểm tra kiểu lúc biên dịch chương trình.
- Khi phối hợp với sự quá tải hàm, quá tải toán tử hoặc con trả hàm ta có thể viết được các chương trình rất hay, ngắn gọn, linh động và có tính tiến hóa cao.



4. Một số dạng hàm đặc biệt

❖ Nạp chồng hàm

- Nhu cầu
 - Thực hiện một công việc với nhiều cách khác nhau.
Nếu các hàm khác tên sẽ khó quản lý.
- Khái niệm nạp chồng/quá tải (overload) hàm
 - Hàm cùng tên nhưng có tham số đầu vào hoặc đầu ra khác nhau.
 - Nguyên mẫu hàm (prototype) khi bỏ tên tham số phải khác nhau.
 - Cho phép người dùng chọn phương pháp thuận lợi nhất để thực hiện công việc.





4. Một số dạng hàm đặc biệt

❖ Ví dụ

- Nhập mảng theo nhiều cách

```
void Nhap(int a[], int &n)
{
    // Nhập n rồi nhập mảng a
}
void Nhap(int a[], int n)
{
    // Nhập mảng a theo n truyền vào
}
int Nhap(int a[])
{
    // Nhập n, nhập mảng a rồi trả n về
}
```

NTU - Kỹ thuật lập trình



4. Một số dạng hàm đặc biệt

❖ Nạp chồng toán tử

- Khái niệm

- Giống như quá tải hàm.
- Có một số quy định khác về tham số.
- Tạo thêm hàm toán tử (operator function) để nạp chồng toán tử.

```
<kiểu trả về> operator#(<ds tham số>
{
    // Các thao tác cần thực hiện
}
```

- # là toán tử (trừ . :: .* ?), <ds tham số> phụ thuộc vào toán tử được nạp chồng.

NTU - Kỹ thuật lập trình





4. Một số dạng hàm đặc biệt

❖ Ví dụ

- Toán tử + (cho hai PHANSO)

```
typedef struct {int tu, mau;} PHANSO;

PHANSO operator+(PHANSO ps1, PHANSO ps2)
{
    PHANSO ps;
    ps.tu = ps1.tu*ps2.mau + ps2.tu*ps1.mau;
    ps.mau = ps1.mau*ps2.mau;
    return ps;
}
...
PHANSO a = {1, 2}, b = {3, 4}, c = {5, 6};
PHANSO d = a + b + c; // ⇔ d = a + (b + c)
```

NTU - Kỹ thuật lập trình



5. Bài tập Chương 3

- Cho 2 số nguyên a, b. Viết hàm hoán vị giá trị 2 số trên.
- Viết chương trình nhập số nguyên dương n gồm 5 chữ số, kiểm tra xem các chữ số n có phải là số đối xứng hay không.
- Viết chương trình nhập số nguyên dương n gồm k chữ số($0 < k \leq 5$) , đếm xem n có bao nhiêu chữ số chẵn và bao nhiêu chữ số lẻ.
- Viết chương trình nhập số nguyên dương n gồm k chữ số($0 < k \leq 5$) , đếm xem n có bao nhiêu chữ số là số nguyên tố.
- Viết chương trình nhập số nguyên dương n gồm k chữ số($0 < k \leq 5$) , tính tổng các ước số dương của n.

NTU - Kỹ thuật lập trình





5. Bài tập Chương 3

1. Cho 2 số nguyên a, b. Viết hàm hoán vị giá trị 2 số trên.
2. Viết chương trình nhập số nguyên dương n gồm 5 chữ số, kiểm tra xem các chữ số n có phải là số đối xứng hay không.
3. Viết chương trình nhập số nguyên dương n gồm k chữ số($0 < k \leq 5$) , đếm xem n có bao nhiêu chữ số chẵn và bao nhiêu chữ số lẻ.
4. Viết chương trình nhập số nguyên dương n gồm k chữ số($0 < k \leq 5$) , đếm xem n có bao nhiêu chữ số là số nguyên tố.
5. Viết chương trình nhập số nguyên dương n gồm k chữ số($0 < k \leq 5$) , tính tổng các ước số dương của n.



NTU - Kỹ thuật lập trình



5. Bài tập Chương 3

6. Cho biết kết quả đoạn CT sau và giải thích

```
int divide (int a, int b)
{
    return (a/b);
}
float divide (float a, float b)
{
    return (a/b);
}
void main ()
{
    int x=5,y=2; float n=5.0, m=2.0;
    cout<<divide (x,y); cout<<"\n"; cout<<divide (n,m);
}
```



NTU - Kỹ thuật lập trình



Chương 4

Các cấu trúc dữ liệu cơ bản

Bài giảng Kỹ thuật lập trình



Nội dung

- 1. Đặt vấn đề
- 2. Kiểu mảng
- 3. Kiểu chuỗi
- 4. Kiểu con trỏ
- 5. Bài tập Chương 4



1. Đặt vấn đề

- ❖ Các KDL cơ sở như char, int, long, float... được sử dụng để khai báo biến đơn giản.

- ❖ Xét ví dụ sau:

- Viết chương trình thực hiện các công việc:
 - Nhập vào dãy N số nguyên
 - Tính tổng dãy số
 - Tính tổng số đầu tiên và số cuối cùng
 - Sắp xếp dãy số tăng dần theo giá trị

Sử dụng các KDL cài đặt ví dụ trên với các trường hợp:

- N = 3
- N = 30
- N nhập từ bàn phím

NTU - Kỹ thuật lập trình



2. Kiểu mảng

- ❖ Mảng là một tổ chức kiểu dữ liệu gồm một số cố định các phần tử có cùng kiểu.

- ❖ **Khai báo mảng tường minh**

- <kiểu cơ sở> <tên biến mảng> [<số phần tử>];**
- <kiểu cơ sở> <tên biến mảng> [<N1>] [<N2>] ... [<Nn>];**
- <N1>, ..., <Nn> : số lượng phần tử của mỗi chiều.

- ❖ **Lưu ý**

- Phải xác định **<số phần tử>** cụ thể (**hằng**) khi khai báo.
- Mảng nhiều chiều: **<tổng số phần tử> = N1*N2*...*Nn**
- Bộ nhớ sử dụng = **<tổng số phần tử> * sizeof(<kiểu cơ sở>)**
- Bộ nhớ sử dụng phải **ít hơn 64KB** (65536 B)
- Một dãy liên tục có chỉ số từ **0** đến **<số phần tử> - 1**

NTU - Kỹ thuật lập trình





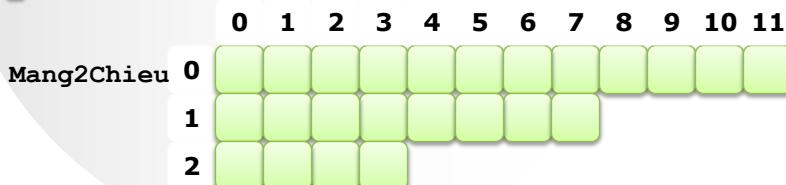
2. Kiểu mảng

❖ Ví dụ

```
int Mang1Chieu[10];
```



```
int Mang2Chieu[3][4];
```



NTU - Kỹ thuật lập trình



2. Kiểu mảng

❖ Nhập mảng

```
void NhapMang(int a[], int &n)
{
    printf("Nhập số lượng phần tử n: ");
    scanf("%d", &n);

    for (int i = 0; i < n; i++)
    {
        printf("Nhập phần tử thứ %d: ", i);
        scanf("%d", &a[i]);
    }
}
```

NTU - Kỹ thuật lập trình





2. Kiểu mảng

❖ Xuất mảng

```
void XuatMang(int a[], int n)
{
    printf("Noi dung cua mang la: ");

    for (int i = 0; i < n; i++)
        printf("%d ", a[i]);

    printf("\n");
}
```

NTU - Kỹ thuật lập trình



2. Kiểu mảng

❖ Bài toán tìm kiếm một phần tử trong mảng

- Tìm xem phần tử **x** có nằm trong mảng **a** kích thước **n** hay không? Nếu có thì nó nằm ở vị trí đầu tiên nào.

❖ Ý tưởng (Tìm kiếm tuần tự/tuyến tính)

- Xét từng phần tử của mảng **a**, bắt đầu từ vị trí **0**. Nếu phần tử đang xét bằng **x** thì trả về vị trí đó. Nếu không tìm được thì trả về **-1**.

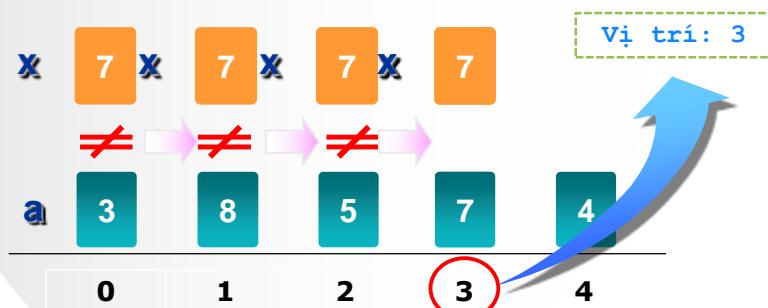
NTU - Kỹ thuật lập trình





2. Kiểu mảng

❖ Ví dụ minh họa



NTU - Kỹ thuật lập trình



2. Kiểu mảng

❖ Hàm tìm kiếm (1)

```
int TimKiem(int a[], int n, int x)
{
    int vt = 0;

    while (vt < n && a[vt] != x)
        vt++;

    if (vt < n)
        return vt;
    else
        return -1;
}
```

NTU - Kỹ thuật lập trình





2. Kiểu mảng

❖ Hàm tìm kiếm (2)

```
int TimKiem(int a[], int n, int x)
{
    for (int vt = 0; vt < n; vt++)
        if (a[vt] == x)
            return vt;

    return -1;
}
```

NTU - Kỹ thuật lập trình



2. Kiểu mảng

❖ Bài toán tìm giá trị lớn nhất

- Cho trước mảng **a** có **n** phần tử. Tìm giá trị lớn nhất trong **a** (là **max**)

❖ Ý tưởng

- Giả sử giá trị **max hiện tại** là giá trị phần tử đầu tiên **a[0]**
- Lần lượt kiểm tra các phần tử còn lại để cập nhật **max**.

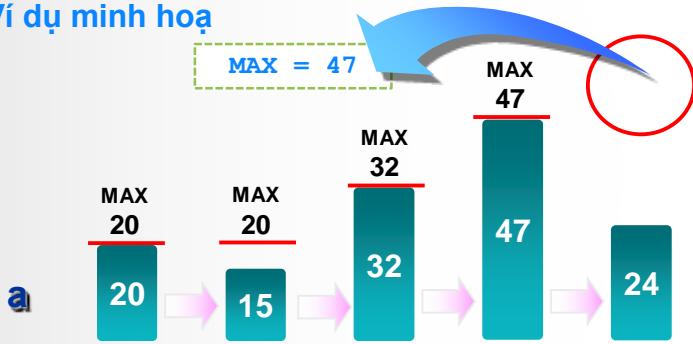
NTU - Kỹ thuật lập trình





2. Kiểu mảng

❖ Ví dụ minh họa



NTU - Kỹ thuật lập trình



2. Kiểu mảng

❖ Hàm tìm max

```
int TimMax(int a[], int n)
{
    int max = a[0];

    for (int i = 1; i < n; i++)
        if (a[i] > max)
            max = a[i];

    return max;
}
```

NTU - Kỹ thuật lập trình





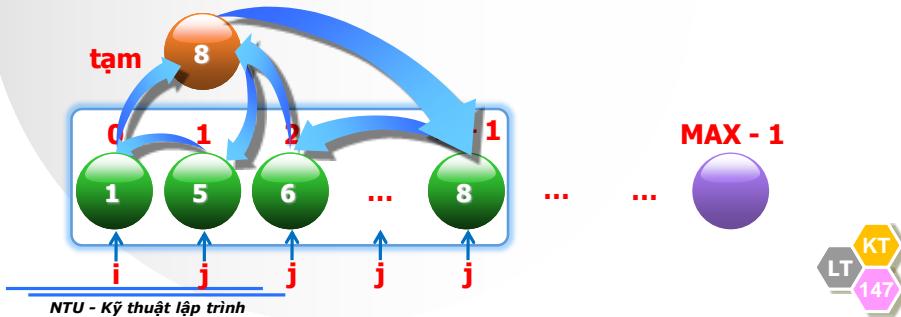
2. Kiểu mảng

❖ Bài toán sắp xếp

- Cho trước mảng a kích thước n . Hãy sắp xếp mảng a đó sao cho các phần tử có giá trị **tăng dần**.

❖ Ý tưởng (PP đổi chỗ trực tiếp)

- Sử dụng 2 biến i và j để so sánh tất cả cặp phần tử với nhau và hoán vị các cặp **nghịch thế** (sai thứ tự).



2. Kiểu mảng

❖ Hàm sắp xếp

```
void SapXepTang(int a[], int n)
{
    int i, j;

    for (i = 0; i < n - 1; i++)
    {
        for (j = i + 1; j < n; j++)
        {
            if (a[i] > a[j])
                HoanVi(a[i], a[j]);
        }
    }
}
```

NTU - Kỹ thuật lập trình





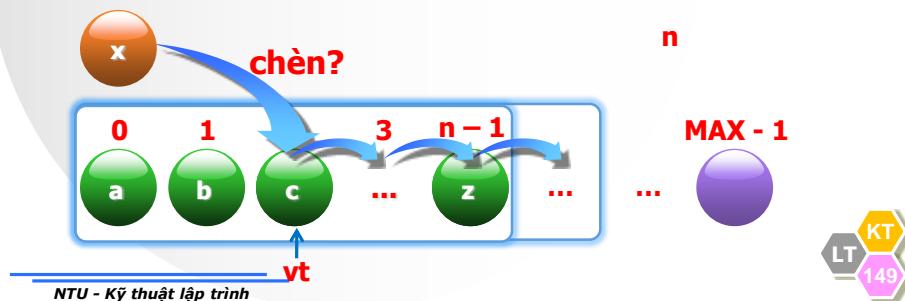
2. Kiểu mảng

❖ Bài toán thêm một phần tử

- Thêm phần tử x vào mảng a kích thước n tại vị trí vt .

❖ Ý tưởng

- Đẩy các phần tử bắt đầu tại vị trí vt sang phải 1 vị trí.
- Đưa x vào vị trí vt trong mảng.
- Tăng n lên 1 đơn vị.



NTU - Kỹ thuật lập trình



2. Kiểu mảng

❖ Hàm thêm phần tử

```
void Them(int a[], int &n, int vt, int x)
{
    if (vt >= 0 && vt <= n)
    {
        for (int i = n; i > vt; i--)
            a[i] = a[i - 1];

        a[vt] = x;
        n++;
    }
}
```

NTU - Kỹ thuật lập trình





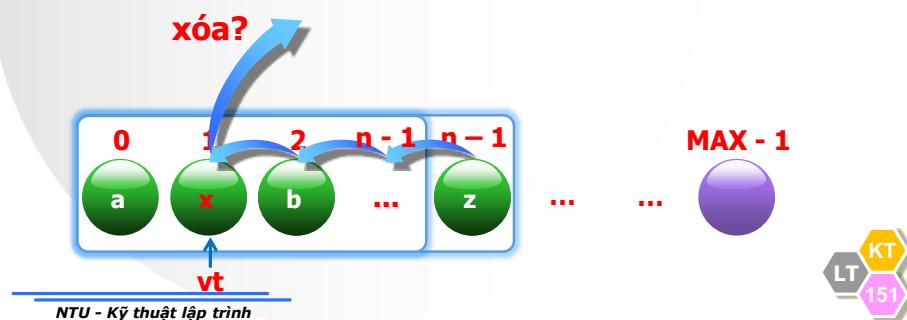
2. Kiểu mảng

❖ Bài toán xóa một phần tử

- Xóa một phần tử trong mảng a kích thước n tại vị trí vt

❖ Ý tưởng

- “Kéo” các phần tử bên phải vị trí vt sang trái 1 vị trí.
- Giảm n xuống 1 đơn vị.



2. Kiểu mảng

❖ Hàm xóa một phần tử

```
void Xoa(int a[], int &n, int vt)
{
    if (vt >= 0 && vt < n)
    {
        for (int i = vt; i < n - 1; i++)
            a[i] = a[i + 1];
        n--;
    }
}
```

NTU - Kỹ thuật lập trình





3. Kiểu chuỗi

❖ Khái niệm

- Là **trường hợp đặc biệt** của kiểu mảng các ký tự.
- Chuỗi ký tự **kết thúc bằng ký tự có mã ASCII là 0 ('0')**
- ➔ Độ dài chuỗi = kích thước mảng – 1
- Khi áp dụng các phép toán trên chuỗi, ta sử dụng **các hàm trong string.h**

❖ Ví dụ

char s[10];



NTU - Kỹ thuật lập trình



3. Kiểu chuỗi

❖ Khởi tạo như mảng thông thường

- Độ dài cụ thể

```
char s[10] = {'D', 'A', ' ', 'L', 'A', 'T', '\0'};  
char s[10] = "DA LAT"; // Tự động thêm '\0'
```



- Tự xác định độ dài

```
char s[] = {'D', 'A', ' ', 'L', 'A', 'T', '\0'};  
char s[] = "DA LAT"; // Tự động thêm '\0'  
0 1 2 3 4 5 6
```



NTU - Kỹ thuật lập trình





3. Kiểu chuỗi

➤ Xuất chuỗi

❖ Sử dụng hàm printf với đặc tả "%s"

```
char hocphan[50] = "Lap Trinh C/C++";
printf("%s", hocphan); // Không xuống dòng
```

Lap Trinh C/C++ _

❖ Sử dụng hàm puts

```
char hocphan[50] = "Lap Trinh C/C++";
puts(hocphan); // Tự động xuống dòng
⇒ printf("%s\n", hocphan);
```

Lap Trinh C/C++

_

NTU - Kỹ thuật lập trình



3. Kiểu chuỗi

➤ Nhập chuỗi

❖ Sử dụng hàm scanf với đặc tả "%s"

- Chỉ nhận các ký tự từ bàn phím đến khi gặp ký tự khoảng trắng hoặc ký tự xuống dòng.
- Chuỗi nhận được không bao gồm ký tự khoảng trắng và xuống dòng.

```
char hocphan[50];
printf("Nhập một chuỗi: ");
scanf("%s", hocphan);
printf("Chuỗi nhận được là: %s", hocphan);
```

Nhập một chuỗi: Lap Trinh C/C++
Chuỗi nhận được là: Lap_

NTU - Kỹ thuật lập trình





3. Kiểu chuỗi

➤ Nhập chuỗi

❖ Sử dụng hàm gets

- Nhận các ký tự từ bàn phím đến khi gặp ký tự xuống dòng.
- Chuỗi nhận được là những gì người dùng nhập (trừ ký tự xuống dòng).

```
char hocphan[50];
printf("Nhập một chuỗi: ");
gets(hocphan);
printf("Chuỗi nhận được là: %s", hocphan);
```

```
Nhập một chuỗi: Lập trình C/C++
Chuỗi nhận được là: Lập trình C/C++ _
```

NTU - Kỹ thuật lập trình



3. Kiểu chuỗi

➤ Các hàm trong thư viện string.h

❖ int strlen(char s[]):

Trả về độ dài của chuỗi s, chính là chỉ số của ký tự NUL trong chuỗi.

❖ strcpy(char dest[], char source[]):

Sao chép nội dung chuỗi source vào chuỗi dest.

❖ strncpy(char dest[], char source[], int n):

Tương tự như strcpy(), nhưng ngừng sao chép sau n ký tự. Trường hợp không có đủ số ký tự trong source thì hàm sẽ điền thêm các ký tự trắng vào chuỗi dest.

❖ strcat(char ch1[], char ch2[]):

Nối chuỗi ch2 vào cuối chuỗi ch1. Sau lời gọi hàm này độ dài chuỗi ch1 bằng tổng độ dài của cả hai chuỗi.

NTU - Kỹ thuật lập trình





3. Kiểu chuỗi

❖ **strncat(char ch1[], char ch2[], int n):**

Tương tự như strcat nhưng chỉ giới hạn với n ký tự đầu tiên của ch2

❖ **int strcmp(char ch1[], char ch2[]):**

So sánh hai chuỗi ch1 và ch2. Nguyên tắc so sánh theo kiểu từ điển. Giá trị trả về:

0 nếu chuỗi ch1 bằng chuỗi ch2

>0 nếu chuỗi ch1 lớn hơn chuỗi ch2

<0 nếu chuỗi ch1 nhỏ hơn chuỗi ch2

❖ **int strncmp(char ch1[], char ch2[],int n):**

Tương tự như hàm strcmp(), nhưng chỉ giới hạn việc so sánh với n ký tự đầu tiên của hai chuỗi.



3. Kiểu chuỗi

❖ **int strcasecmp(char ch1[], char ch2[],int n):**

Tương tự như strcasecmp(), nhưng việc so sánh chỉ giới hạn ở n ký tự đầu tiên của mỗi chuỗi.

❖ **char *strchr(char s[], char c):**

Tìm lần xuất hiện đầu tiên của ký tự c trong chuỗi s, trả về địa chỉ của ký tự này.

❖ **char *strrchr(char s[],char c):**

Tương tự như hàm strrchr(), nhưng việc tìm kiếm bắt đầu từ cuối chuỗi.

❖ **strlwr(char s[]):**

Chuyển đổi các chữ in trong chuỗi s sang chữ thường.





3. Kiểu chuỗi

❖ **strupr(char s[]):**

Ngược lại với hàm strlwr()

❖ **strset(char s[], char c):**

Khởi đầu tất cả các ký tự của s bằng ký tự c

❖ **strnset(char s[], char c, int n):**

Khởi đầu giá trị cho n ký tự đầu tiên của s bằng ký tự c

❖ **char *strstr(char s1[], char s2[]):**

Tìm kiếm chuỗi s2 trong chuỗi s1, Trả về địa chỉ của lần xuất hiện đầu tiên của s2 trong s1 hoặc NULL khi không tìm thấy.



4. Kiểu con trỏ

❖ **Bộ nhớ máy tính**

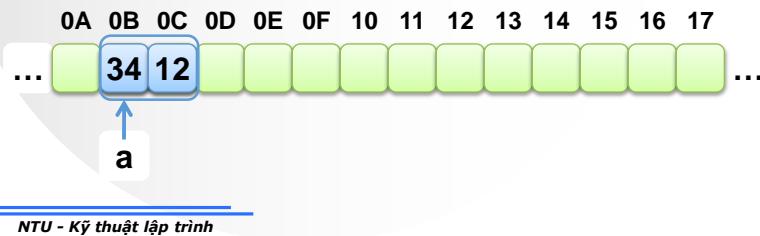
- Bộ nhớ RAM chứa rất **nhiều ô nhớ**, mỗi ô nhớ có **kích thước 1 byte**.
- RAM dùng để chứa **một phần hệ điều hành, các lệnh chương trình, các dữ liệu...**
- Mỗi ô nhớ có **địa chỉ duy nhất** và địa chỉ này được **đánh số từ 0 trở đi**.
- Ví dụ
 - RAM **512MB** được đánh địa chỉ từ **0** đến $2^{29} - 1$
 - RAM **2GB** được đánh địa chỉ từ **0** đến $2^{31} - 1$





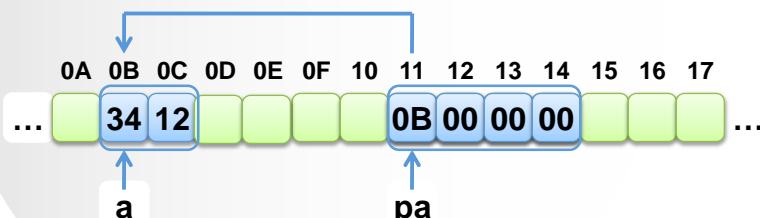
4. Kiểu con trỏ

- ❖ Quy trình xử lý của trình biên dịch
 - Dành riêng một vùng nhớ với **địa chỉ duy nhất** để lưu biến đó.
 - **Liên kết** địa chỉ ô nhớ đó với tên biến.
 - Khi gọi tên biến, nó sẽ **truy xuất tự động** đến ô nhớ đã liên kết với tên biến.
- ❖ Ví dụ: int a = 0x1234; // Giả sử địa chỉ 0x0B



4. Kiểu con trỏ

- ❖ Khái niệm
 - Địa chỉ của biến là một con số.
 - Ta có thể tạo **biến khác để lưu địa chỉ của biến này** → Con trỏ.





4. Kiểu con trỏ

❖ Khai báo

- Giống như mọi biến khác, biến con trỏ muốn sử dụng cũng cần phải được khai báo

<kiểu dữ liệu> *<tên biến con trỏ>;

❖ Ví dụ

```
char *ch1, *ch2;
int *p1, p2;
```

- ch1 và ch2 là biến con trỏ, trỏ tới vùng nhớ kiểu char (1 byte).
- p1 là biến con trỏ, trỏ tới vùng nhớ kiểu int (4 bytes) còn p2 là biến kiểu int bình thường.

NTU - Kỹ thuật lập trình



4. Kiểu con trỏ

❖ Con trỏ NULL

- Con trỏ **NULL** là con trỏ không trỏ và đâu cả.
- Khác với con trỏ chưa được khởi tạo.

```
int n;
int *p1 = &n;
int *p2;      // unreferenced local variable
int *p3 = NULL;
```



NULL

NTU - Kỹ thuật lập trình





4. Kiểu con trỏ

❖ Khởi tạo

- Khi mới khai báo, biến con trỏ được **đặt ở địa chỉ nào đó** (Không biết trước).
 - chứa **giá trị không xác định**
 - **trỏ đến vùng nhớ không biết trước.**
- Đặt địa chỉ của biến vào con trỏ (toán tử **&**)

❖ Ví dụ

```
<tên biến con trỏ> = &<tên biến>;
```

```
int a, b;
int *pa = &a, *pb;
pb = &b;
```

NTU - Kỹ thuật lập trình



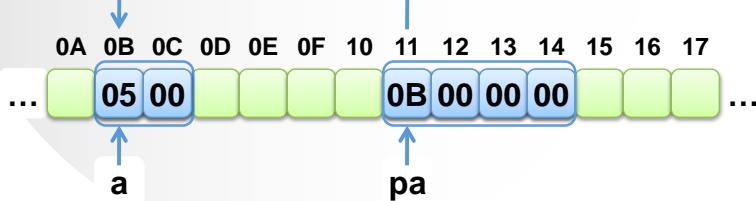
4. Kiểu con trỏ

❖ Truy xuất đến ô nhớ mà con trỏ trỏ đến

- Con trỏ chứa **một số nguyên chỉ địa chỉ**.
- Vùng nhớ mà nó trỏ đến, sử dụng toán tử *****.

❖ Ví dụ

```
int a = 5, *pa = &a;
printf("%d\n", pa); // Giá trị biến pa
printf("%d\n", *pa); // Giá trị vùng nhớ pa trỏ đến
printf("%d\n", &pa); // Địa chỉ biến pa
```



NTU - Kỹ thuật lập trình





4. Kiểu con trỏ

❖ Kích thước của con trỏ

```
char *p1;
int *p2;
float *p3;
double *p4;
...
```

- Con trỏ **chỉ lưu địa chỉ** nên kích thước của mọi con trỏ là như nhau:
 - Môi trường MD-DOS (**16 bit**): **2 bytes** (64KB)
 - Môi trường Windows (**32 bit**): **4 bytes** (4GB)



4. Kiểu con trỏ

❖ Một số lưu ý

- Con trỏ là khái niệm quan trọng và khó nhất trong C. Mức độ thành thạo C được đánh giá qua mức độ sử dụng con trỏ.
- Nắm rõ quy tắc sau, ví dụ **int a, *pa = &a;**
 - ***pa** và **a** đều chỉ **nội dung** của biến **a**.
 - **pa** và **&a** đều chỉ **địa chỉ** của biến **a**.
- **Không nên** sử dụng con trỏ khi chưa được khởi tạo. Kết quả sẽ không lường trước được.

```
int *pa; *pa = 1904; // !!!
```



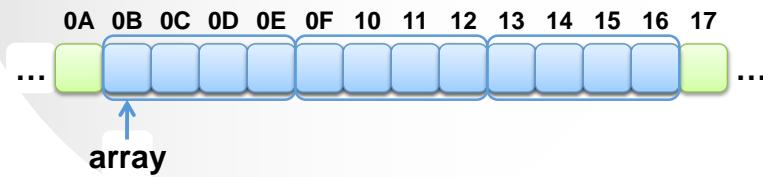


4. Kiểu con trỏ

- ❖ Con trỏ và mảng một chiều

`int array[3];`

- Tên mảng array là một **hằng con trỏ**
→ **không thể thay đổi** giá trị của hằng này.
- Giá trị của array là địa chỉ phần tử đầu tiên của mảng
→ `array == &array[0]`



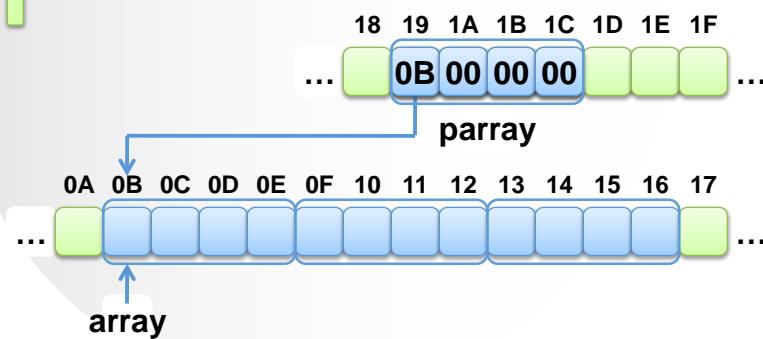
NTU - Kỹ thuật lập trình



4. Kiểu con trỏ

- ❖ Con trỏ đến mảng một chiều

```
int array[3], *parray;
parray = array;           // Cách 1
parray = &array[0];        // Cách 2
```



NTU - Kỹ thuật lập trình

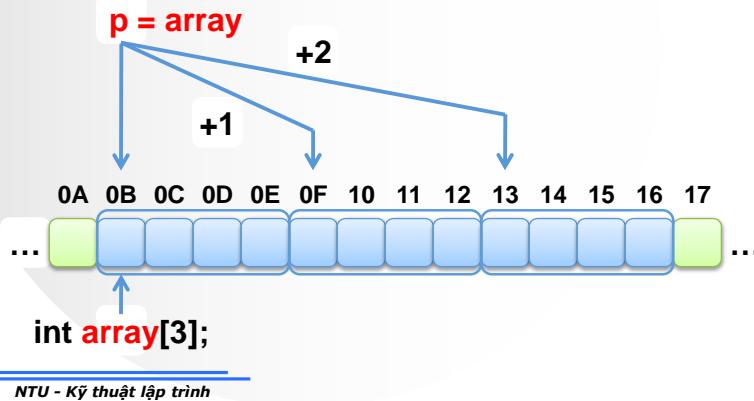




4. Kiểu con trỏ

❖ Phép cộng (tăng)

- $+ n \Leftrightarrow + n * \text{sizeof}(<\text{kiểu dữ liệu}>)$
- Có thể sử dụng toán tử gộp $+=$ hoặc $++$



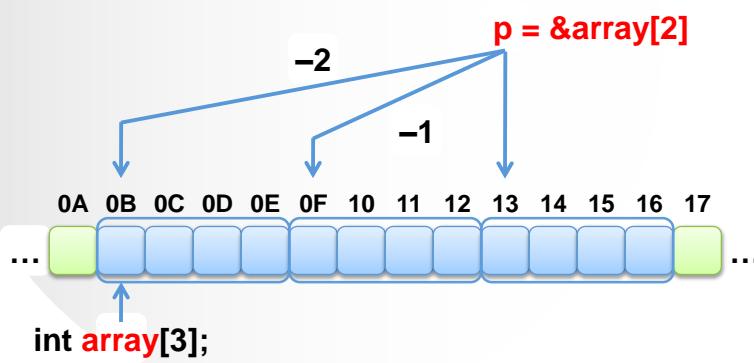
NTU - Kỹ thuật lập trình



4. Kiểu con trỏ

❖ Phép trừ (giảm)

- $- n \Leftrightarrow - n * \text{sizeof}(<\text{kiểu dữ liệu}>)$
- Có thể sử dụng toán tử gộp $--$ hoặc $--$

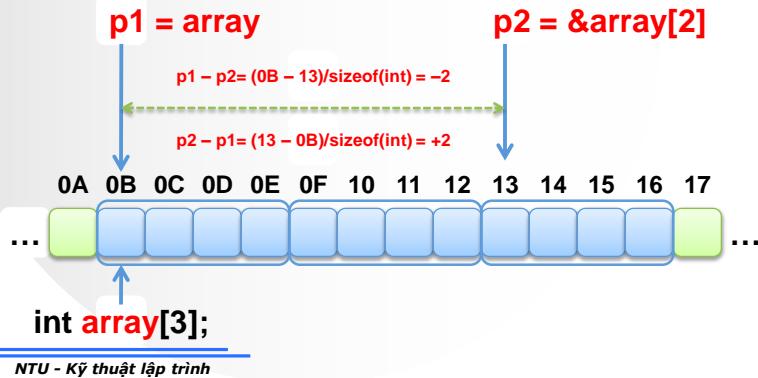


NTU - Kỹ thuật lập trình



4. Kiểu con trỏ

- ❖ Phép toán tính khoảng cách giữa 2 con trỏ
 - <kiểu dữ liệu> *p1, *p2;
 - p1 – p2 cho ta khoảng cách (theo số phần tử) giữa hai con trỏ (cùng kiểu)



4. Kiểu con trỏ

- ❖ Các phép toán khác
 - Phép so sánh: So sánh địa chỉ giữa hai con trỏ (thứ tự ô nhớ)
 - == !=
 - > >=
 - < <=
 - Không thể thực hiện các phép toán: * / %

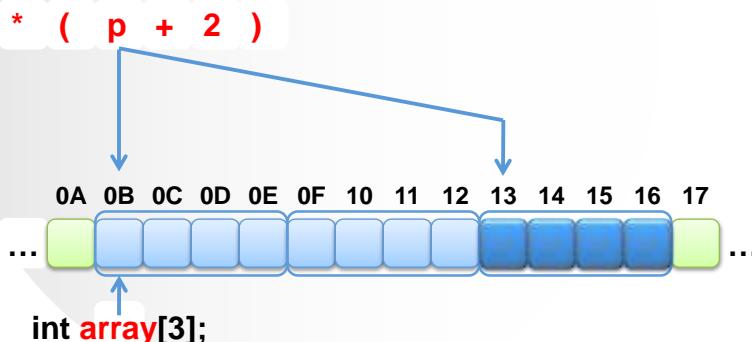




4. Kiểu con trỏ

- ❖ Truy xuất đến phần tử thứ n của mảng

- int array[3], n = 2, *p = array;
- $\rightarrow \text{array}[n] == p[n] == *(p + n)$



NTU - Kỹ thuật lập trình



4. Kiểu con trỏ

- ❖ Ví dụ nhập mảng

```
void main()
{
    int a[10], n = 10, *pa;
    pa = a; // hoặc pa = &a[0];

    for (int i = 0; i < n; i++)
        scanf("%d", &a[i]);
        scanf("%d", &pa[i]);
        scanf("%d", a + i);
        scanf("%d", pa + i);
        scanf("%d", a++);
        scanf("%d", pa++);
}
```

$\Rightarrow \&a[i] \Leftrightarrow (a + i) \Leftrightarrow (pa + i) \Leftrightarrow \&pa[i]$

NTU - Kỹ thuật lập trình





4. Kiểu con trỏ

❖ Ví dụ xuất mảng

```
void main()
{
    int a[10], n = 10, *pa;
    pa = a;          // hoặc pa = &a[0];
    ...
    for (int i = 0; i<n; i++)
        printf("%d", a[i]);
        printf("%d", pa[i]);
        printf("%d", *(a + i));
        printf("%d", *(pa + i));
        printf("%d", *(a++));
        printf("%d", *(pa++));
}
```

→ $a[i] \Leftrightarrow *(a + i) \Leftrightarrow *(pa + i) \Leftrightarrow pa[i]$

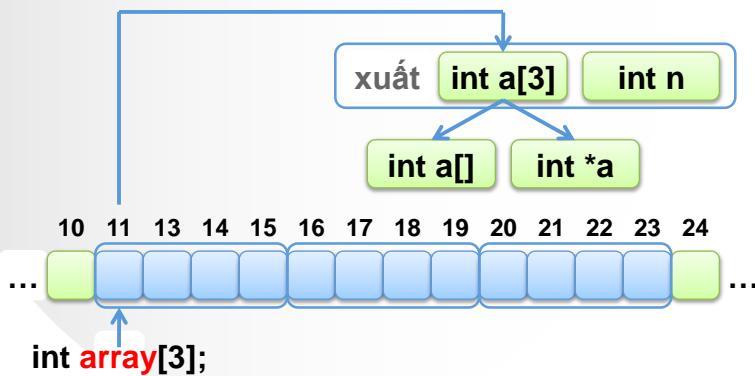
NTU - Kỹ thuật lập trình



4. Kiểu con trỏ

❖ Chú ý!

- Mảng một chiều truyền cho hàm là **địa chỉ của phần tử đầu tiên** chứ không phải toàn mảng.



NTU - Kỹ thuật lập trình





4. Kiểu con trỏ

❖ Ví dụ

```
void xuat(int a[10], int n)
{
    for (int i = 0; i<n; i++)
        printf("%d", *(a++)); // OK
}
void main()
{
    int a[10], n = 10;

    for (int i = 0; i<n; i++)
        printf("%d", *(a++)); // Lỗi
}
```

→ Đôi số mảng truyền cho hàm **không phải hằng con trỏ**.



4. Kiểu con trỏ

❖ Lưu ý

- **Không** thực hiện các phép toán *, /, %.
- Tăng/giảm con trỏ n đơn vị có nghĩa là tăng/giảm giá trị của nó $n * \text{sizeof}(<\text{kiểu dữ liệu mà nó trả về}>)$ (bytes)
- **Không thể** tăng/giảm biến mảng (con trỏ hằng). Hãy gán một con trỏ đến địa chỉ đầu của mảng và tăng/giảm con trỏ đó.
- **Đối** số mảng một chiều truyền cho hàm là địa chỉ phần tử đầu tiên của mảng.





4. Kiểu con trỏ

❖ Con trỏ cấp 2

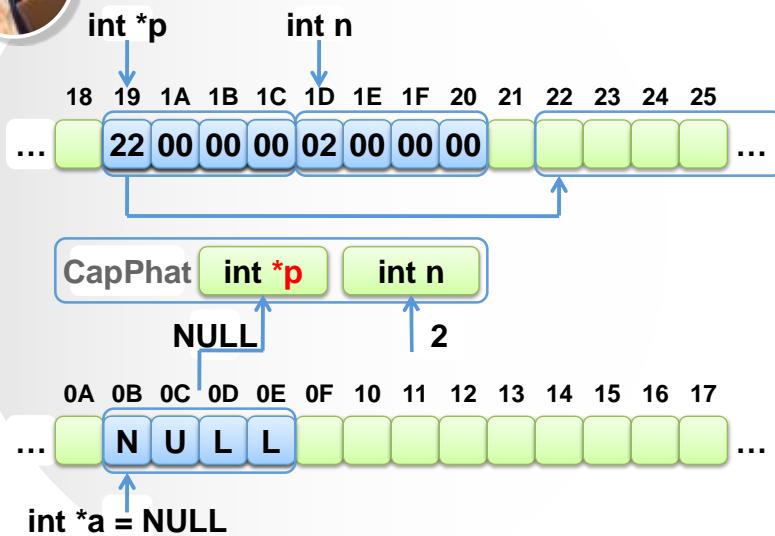
```
void CapPhat(int *p, int n)
{
    p = (int *)malloc(n * sizeof(int));
}
void main()
{
    int *a = NULL;
    CapPhat(a, 2);
    // a vẫn = NULL
}
```

Làm sao thay đổi giá trị của con trỏ (không phải giá trị mà nó trỏ đến) sau khi gọi hàm?

NTU - Kỹ thuật lập trình



4. Kiểu con trỏ



NTU - Kỹ thuật lập trình





4. Kiểu con trỏ

❖ Giải pháp

- Sử dụng tham chiếu `int *&p` (trong C++)

```
void CapPhat(int *&p, int n)
{
    p = (int *)malloc(n * sizeof(int));
}
```

- Không thay đổi trực tiếp tham số mà trả về

```
int* CapPhat(int n)
{
    int *p = (int *)malloc(n * sizeof(int));
    return p;
}
```

NTU - Kỹ thuật lập trình



4. Kiểu con trỏ

❖ Giải pháp

- Sử dụng con trỏ p trỏ đến con trỏ a này. Hàm sẽ thay đổi giá trị của con trỏ a gián tiếp thông qua con trỏ p.

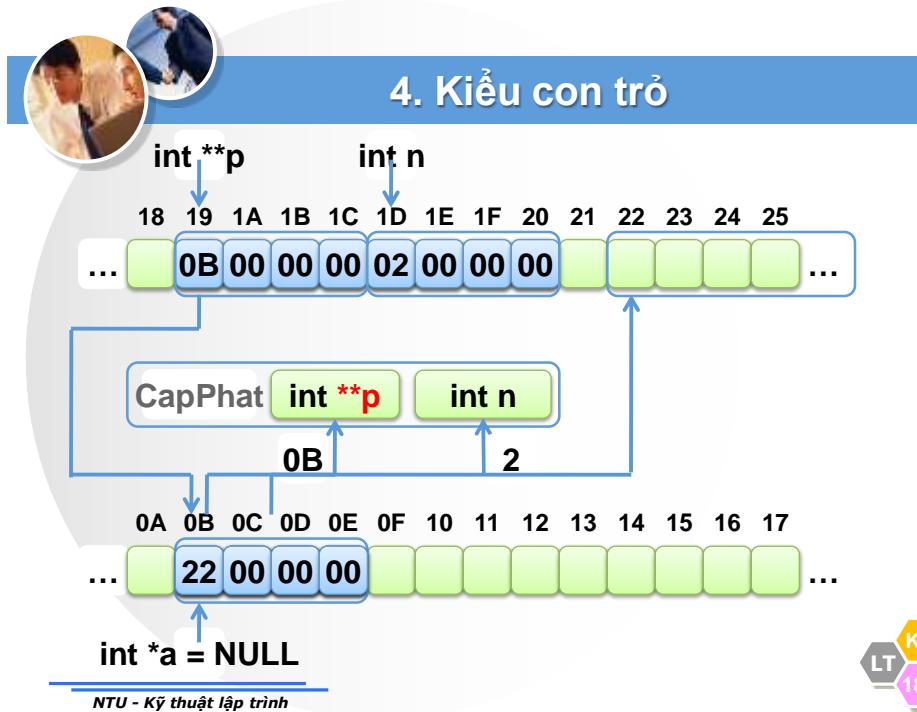
```
void CapPhat(int **p, int n)
{
    *p = (int *)malloc(n * sizeof(int));
}

void main()
{
    int *a = NULL;
    CapPhat(&a, 4);
}
```

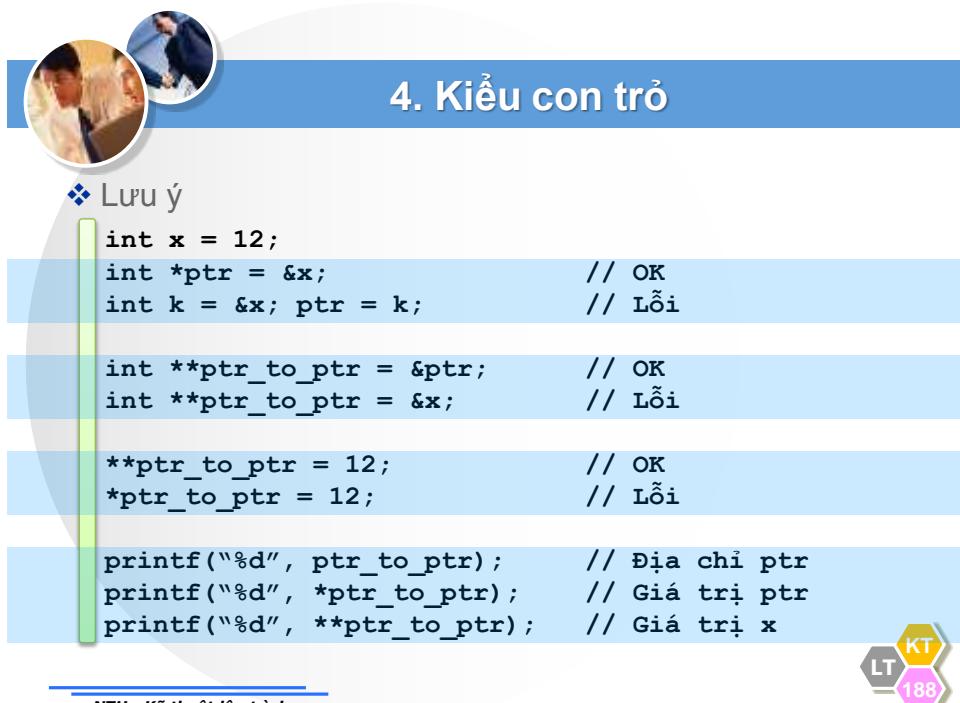
NTU - Kỹ thuật lập trình



4. Kiểu con trỏ



4. Kiểu con trỏ

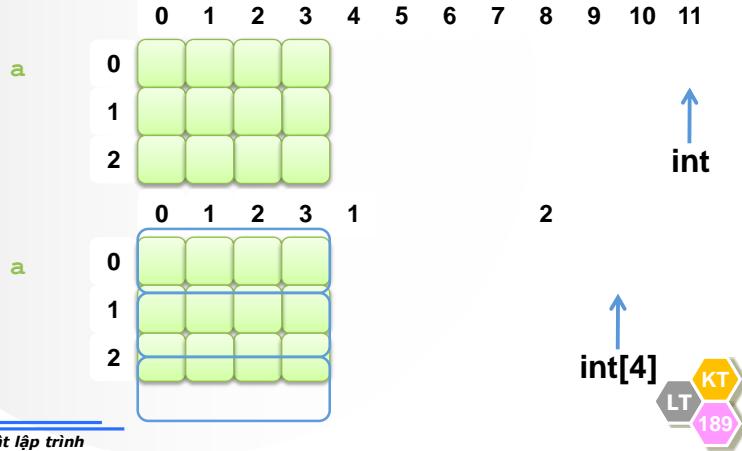




4. Kiểu con trỏ

❖ Con trỏ và mảng 2 chiều

`int a[3][4];`



NTU - Kỹ thuật lập trình



4. Kiểu con trỏ

❖ Hướng tiếp cận 1

- Các phần tử tạo thành mảng 1 chiều
- Sử dụng con trỏ `int *` để duyệt mảng 1 chiều

`int *p = (int *)a`

`+1`



NTU - Kỹ thuật lập trình





4. Kiểu con trỏ

- ❖ Nhập / Xuất theo chỉ số mảng 1 chiều

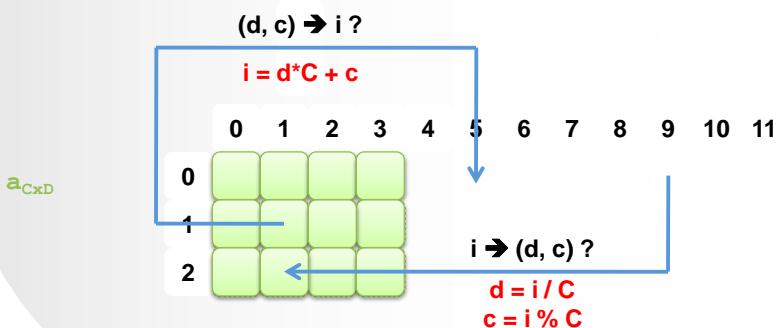
```
#define D 3
#define C 4
void main()
{
    int a[D][C], i;
    int *p = (int *)a;
    for (i = 0; i < D*C; i++)
    {
        printf("Nhập phần tử thứ %d: ", i);
        scanf("%d", p + i);
    }
    for (i = 0; i < D*C; i++)
        printf("%d ", *(p + i));
}
```

NTU - Kỹ thuật lập trình



4. Kiểu con trỏ

- ❖ Liên hệ giữa chỉ số mảng 1 chiều và chỉ số mảng 2 chiều



NTU - Kỹ thuật lập trình





4. Kiểu con trỏ

- ❖ Nhập / Xuất theo chỉ số mảng 2 chiều

```
int a[D][C], i, d, c;
int *p = (int *)a;

for (i = 0; i < D*C; i++)
{
    printf("Nhập a[%d][%d]: ", i / C, i % C);
    scanf("%d", p + i);
}
for (d = 0; d < D; d++)
{
    for (c = 0; c < C; c++)
        printf("%d ", *(p + d * C + c)); // *p++
    printf("\n");
}
```

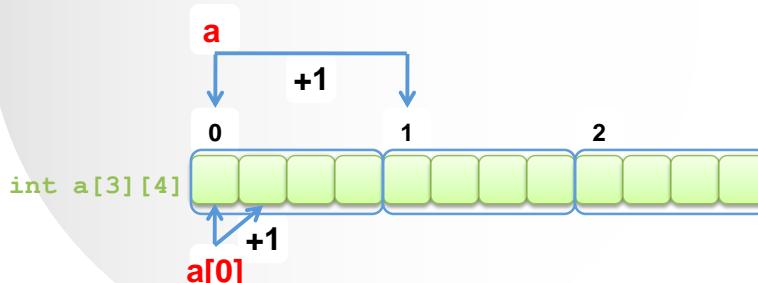
NTU - Kỹ thuật lập trình



4. Kiểu con trỏ

- ❖ Hướng tiếp cận 2

- Mảng 1 chiều, mỗi phần tử là mảng 1 chiều
 - a chứa a[0], a[1], ... \rightarrow a = &a[0]
 - a[0] chứa a[0][0], a[0][1], ... \rightarrow a[0] = &a[0][0]



NTU - Kỹ thuật lập trình

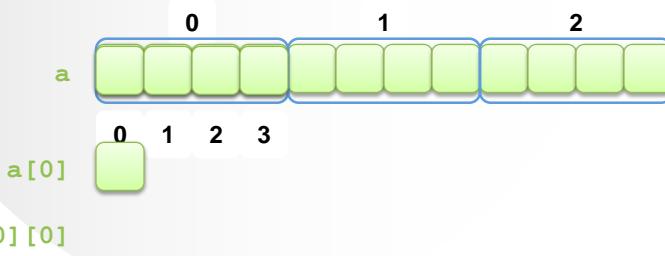




4. Kiểu con trỏ

❖ Kích thước của mảng

```
void main()
{
    int a[3][4];
    printf("KT của a = %d", sizeof(a));
    printf("KT của a[0] = %d", sizeof(a[0]));
    printf("KT của a[0][0] = %d", sizeof(a[0][0]));
}
```



NTU - Kỹ thuật lập trình



4. Kiểu con trỏ

❖ Nhận xét

- a là con trỏ đến a[0], a[0] là con trỏ đến a[0][0] → a là con trỏ cấp 2.
- Có thể truy xuất a[0][0] bằng 3 cách:

```
void main()
{
    int a[3][4];
    a[0][0] = 1;
    *a[0] = 1;
    **a = 1;

    a[1][0] = 1; *a[1] = 1; **(a+1) = 1;
    a[1][2] = 1; *(a[1]+2) = 1; *(*(a+1)+2) = 1;
}
```

NTU - Kỹ thuật lập trình





4. Kiểu con trỏ

❖ Truyền mảng cho hàm

- Truyền địa chỉ phần tử đầu tiên cho hàm.
- Khai báo con trỏ rồi gán địa chỉ mảng cho con trỏ này để nó trỏ đến mảng.
- Con trỏ này phải cùng kiểu với biến mảng, tức là con trỏ đến vùng nhớ n phần tử (mảng)

❖ Cú pháp

❖ Ví dụ

```
<kiểu dữ liệu> (*<tên con trỏ>) [<số phần tử>];
```

```
int (*ptr) [4];
```

NTU - Kỹ thuật lập trình



4. Kiểu con trỏ

❖ Truyền mảng cho hàm

```
void Xuat_1_Mang_C1(int (*ptr) [4]) // ptr[][][4]
{
    int *p = (int *)ptr;
    for (int i = 0; i < 4; i++)
        printf("%d ", *p++);
}

void main()
{
    int a[3][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12}};
    int (*ptr) [4];
    ptr = a;
    for (int i = 0; i < 3; i++)
        Xuat_1_Mang_C1(ptr++); // hoặc ptr + i
        Xuat_1_Mang_C1(a++); // sai => a + 1
}
```

NTU - Kỹ thuật lập trình





4. Kiểu con trỏ

- ❖ Truyền mảng cho hàm

```
void Xuat_1_Mang_C2(int *ptr, int n)          // ptr[]
{
    for (int i = 0; i < n; i++)
        printf("%d ", *ptr++);
}
void main()
{
    int a[3][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12}};
    int (*ptr)[4];
    ptr = a;
    for (int i = 0; i < 3; i++)
        Xuat_1_Mang_C2((int *)ptr++);
        Xuat_1_Mang_C2((int *) (a + i)); // a++ sai
}
```

NTU - Kỹ thuật lập trình



4. Kiểu con trỏ

- ❖ Truyền mảng cho hàm

```
void Xuat_n_Mang_C1(int (*ptr)[4], int n)
{
    int *p = (int *)ptr;
    for (int i = 0; i < n * 4; i++)
        printf("%d ", *p++);
}
void main()
{
    int a[3][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12}};
    int (*ptr)[4];
    ptr = a;

    Xuat_n_Mang_1(ptr, 3);
    Xuat_n_Mang_1(a, 3);
}
```

NTU - Kỹ thuật lập trình





4. Kiểu con trỏ

❖ Truyền mảng cho hàm

```
void Xuat_n_Mang_C2(int (*ptr)[4], int n)
{
    int *p;
    for (int i = 0; i < n; i++)
    {
        p = (int *)ptr++;
        for (int i = 0; i < 4; i++)
            printf("%d ", *p++);
        printf("\n");
    }
}
```

NTU - Kỹ thuật lập trình



4. Kiểu con trỏ

❖ Mảng con trỏ

- Sử dụng cấu trúc dữ liệu nào để lưu trữ thông tin sau?

	0	1	2	3	4	5	6	7
0	1	5	6					
1	2	9	1	2	1	7	0	6
2	0	2						

❖ Giải pháp?

- Cách 1: Mảng 2 chiều 3x8 (tốn bộ nhớ)

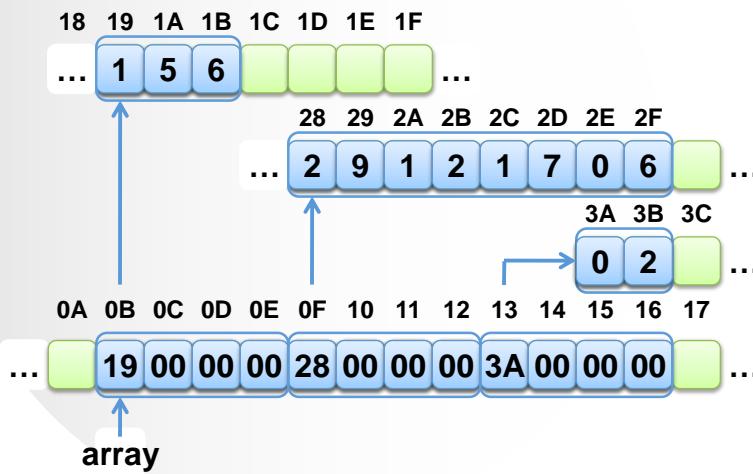
NTU - Kỹ thuật lập trình





4. Kiểu con trỏ

- Cách 2: Mảng 1 chiều các con trỏ



NTU - Kỹ thuật lập trình



4. Kiểu con trỏ

- Ví dụ

```
void print_strings(char *p[], int n)
{
    for (int i = 0; i < n; i++)
        printf("%s ", p[i]);
}

void main()
{
    char *message[4] = {"Dai", "Hoc", "Nha", "Trang"};
    print_strings(message, 4);
}
```

NTU - Kỹ thuật lập trình

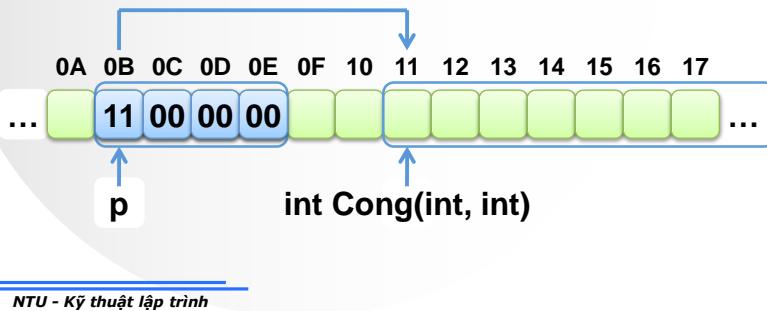




4. Kiểu con trỏ

❖ Con trỏ hàm

- **Hàm** cũng được lưu trữ trong bộ nhớ, tức là **cũng có địa chỉ**.
- Con trỏ hàm là **con trỏ trỏ đến vùng nhớ chứa hàm** và có thể gọi hàm thông qua con trỏ đó.



NTU - Kỹ thuật lập trình



4. Kiểu con trỏ

❖ Khai báo tường minh

```
<kiểu trả về> (* <tên biến con trỏ>) (ds tham số);
```

❖ Ví dụ

```
// Con trỏ đến hàm nhận đối số int, trả về int
int (*ptof1)(int x);
```

```
// Con trỏ đến hàm nhận 2 đối số double, không trả về
void (*ptof2)(double x, double y);
```

```
// Con trỏ đến hàm nhận đối số mảng, trả về char
char (*ptof3)(char *p[]);
```

```
// Con trỏ đến không nhận đối số và không trả về
void (*ptof4)();
```

NTU - Kỹ thuật lập trình





4. Kiểu con trỏ

- ❖ Khai báo không tường minh (thông qua kiểu)

```
typedef <kiểu trả về> (*<tên kiểu>)(ds tham số);  
<tên kiểu> <tên biến con trỏ>;
```

- ❖ Ví dụ

```
int (*pt1)(int, int); // Tường minh  
  
typedef int (*PhépToán)(int, int);  
  
PhépToán pt2, pt3; // Không tường minh
```

NTU - Kỹ thuật lập trình



4. Kiểu con trỏ

- ❖ Gán giá trị cho con trỏ hàm

```
<bien con tro ham> = <tên hàm>;  
<bien con tro ham> = &<tên hàm>;
```

- Hàm được gán phải cùng dạng (vào, ra)

- ❖ Ví dụ

```
int Cong(int x, int y); // Hàm  
int Tru(int x, int y); // Hàm  
int (*tinhtoan)(int x, int y); // Con trỏ hàm  
  
tinhtoan = Cong; // Dạng ngắn gọn  
tinhtoan = &Tru; // Dạng sử dụng địa chỉ  
tinhtoan = NULL; // Không trỏ đến đâu cả
```

NTU - Kỹ thuật lập trình





4. Kiểu con trỏ

- ❖ So sánh con trỏ hàm

```
if  (tinhtoan != NULL)
{
    if (tinhtoan == &Cong)
        printf("Con trỏ đến hàm Cong.");
    else
        if (tinhtoan == &Tru)
            printf("Con trỏ đến hàm Tru.");
        else
            printf("Con trỏ đến hàm khác.");
}
else
    printf("Con trỏ chưa được khởi tạo!");
```

NTU - Kỹ thuật lập trình



4. Kiểu con trỏ

- ❖ Gọi hàm thông qua con trỏ hàm

- Sử dụng toán tử lấy nội dung "*" (chính quy) nhưng trường hợp này có thể bỏ

```
int Cong(int x, int y);
int Tru(int x, int y);

int (*tinhtoan)(int, int);

tinhtoan = Cong;
int kq1 = (*tinhtoan)(1, 2); // Chính quy
int kq2 = tinhtoan(1, 2); // Ngắn gọn
```

NTU - Kỹ thuật lập trình





4. Kiểu con trỏ

- ❖ Truyền tham số là con trỏ hàm

```

int Cong(int x, int y);
int Tru(int x, int y);
int TinhToan(int x, int y, int (*pheptoan)(int, int))
{
    int kq = (*pheptoan)(x, y); // Gọi hàm
    return kq;
}

void main()
{
    int (*pheptoan)(int, int) = &Cong;
    int kq1 = TinhToan(1, 2, pheptoan);
    int kq2 = TinhToan(1, 2, &Tru);
}

```

NTU - Kỹ thuật lập trình



4. Kiểu con trỏ

- ❖ Trả về con trỏ hàm

```

int (*LayPhepToan(char code))(int, int)
{
    if (code == '+')
        return &Cong;
    return &Tru;
}

void main()
{
    int (*pheptoan)(int, int) = NULL;
    pheptoan = LayPhepToan('+');
    int kq2 = pheptoan(1, 2, &Tru);
}

```

NTU - Kỹ thuật lập trình





4. Kiểu con trỏ

- ❖ Trả về con trỏ hàm (khai báo kiểu)

```
typedef (*PhepToan) (int, int);
PhepToan LayPhepToan(char code)
{
    if (code == '+')
        return &Cong;
    return &Tru;
}
void main()
{
    PhepToan pheptoan = NULL;
    pheptoan = LayPhepToan('+');
    int kq2 = pheptoan(1, 2, &Tru);
}
```

NTU - Kỹ thuật lập trình



4. Kiểu con trỏ

- ❖ Mảng con trỏ hàm

```
typedef (*PhepToan) (int, int);
void main()
{
    int (*array1[2])(int, int); // tường minh
    PhepToan array2[2]; // kô tường minh

    array1[0] = array2[1] = &Cong;
    array1[1] = array2[0] = &Tru;

    printf("%d\n", (*array1[0])(1, 2));
    printf("%d\n", array1[1](1, 2));
    printf("%d\n", array2[0](1, 2));
    printf("%d\n", array2[1](1, 2));
}
```

NTU - Kỹ thuật lập trình





4. Kiểu con trỏ

❖ Lưu ý

- Không được quên dấu () khi khai báo con trỏ hàm
 - int (*PhepToan)(int x, int y);
 - int *PhepToan(int x, int y);
- Có thể bỏ tên biến số trong khai báo con trỏ hàm
 - int (*PhepToan)(int x, int y);
 - int (*PhepToan)(int, int);



5. Bài tập Chương 4

1. Viết các hàm thực hiện các chức năng sau:

- Đổi giá trị 2 số nguyên
- Tìm kiếm 1 phần tử trong 1 dãy số
- Sắp xếp dãy số tăng dần sử dụng hàm trả về tham chiếu

2. Cho biết:

- Toán tử nào dùng để xác định địa chỉ của một biến?
- Toán tử nào dùng để xác định giá trị của biến do con trỏ đến?
- 2 cách lấy địa chỉ phần tử đầu tiên của mảng một chiều?
- 6 loại phép toán có thể thực hiện trên con trỏ?





5. Bài tập Chương 4

3. Cho trước:

```
float pay; float *ptr_pay; pay=2313.54; ptr_pay = &pay;
```

Hãy cho biết giá trị của:

- pay
- *ptr_pay
- *pay
- &pay

4. Cho trước:

```
int *pint; float f; char c; double *pd;
```

Hãy chọn phát biểu sai cú pháp:

- f = *pint;
- c = *pd;
- *pint = *pd;
- pd = f;



NTU - Kỹ thuật lập trình



5. Bài tập Chương 4

5. Tìm lỗi trong chương trình sau

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

void main()
{
    int *x, y = 2;
    *x = y;
    *x += y++;
    printf("%d %d", *x, y);
    getch();
}
```



NTU - Kỹ thuật lập trình



5. Bài tập Chương 4

6. Cho con trỏ p1 trỏ đến phần tử thứ 3 còn con trỏ p2 trỏ đến phần tử thứ 4 của mảng int/float thì $p2 - p1 = ?$
7. Gán giá trị 100 cho biến cost sử dụng hai cách trực tiếp và gián tiếp.
8. Sử dụng con trỏ để làm lại các bài tập về mảng một chiều.
 - Nhập/Xuất mảng
 - Tìm phần tử thỏa yêu cầu
 - Tính tổng/đếm các phần tử thỏa yêu cầu
 - Sắp xếp tăng/giảm



5. Bài tập Chương 4

9. Viết đoạn lệnh khai báo biến x kiểu float, khai báo và khởi tạo con trỏ px đến biến x và khai báo và khởi tạo con trỏ ppx đến con trỏ px.
10. Cho biết ý nghĩa của các khai báo sau:
 - int *var1;
 - int var2;
 - int **var3;
11. Cho biết ý nghĩa của các khai báo sau:
 - int a[3][12];
 - int (*b)[12];
 - int *c[12];





5. Bài tập Chương 4

12. Tìm lỗi sai trong đoạn lệnh sau

- int x[3][12];
- int *ptr[12];
- ptr = x;

13. Viết chương trình khai báo mảng hai chiều có 12×12 phần tử kiểu char. Gán ký tự 'X' cho mọi phần tử của mảng này. Sử dụng con trỏ đến mảng để in giá trị các phần tử mảng lên màn hình ở dạng lưới.

14. Viết chương trình khai báo mảng 10 con trỏ đến kiểu float, nhận 10 số thực từ bàn phím, sắp xếp lại và in ra màn hình dãy số đã sắp xếp.

NTU - Kỹ thuật lập trình



Chương 5 Kiểu dữ liệu cấu trúc

Bài giảng Kỹ thuật lập trình



Nội dung

1. Khái niệm kiểu cấu trúc

2. Khai báo

3. Một số cấu trúc phức tạp

4. Kiểu union

Bài tập Chương 5

NTU - Kỹ thuật lập trình



1. Khái niệm kiểu cấu trúc

❖ Khái niệm

- Là kiểu dữ liệu bao gồm **nhiều thành phần có kiểu khác nhau**, mỗi thành phần được gọi là một trường (field).
- Các thành phần được **truy nhập thông qua tên**.
- Khái niệm cấu trúc trong C/C++ có nhiều nét tương tự như khái niệm về bản ghi (record) trong PASCAL

NTU - Kỹ thuật lập trình





1. Khái niệm kiểu cấu trúc

❖ Ví dụ

▪ Thông tin 1 SV

- MSSV : kiểu chuỗi
- Tên SV : kiểu chuỗi
- NTNS : kiểu chuỗi
- Phái : kiểu ký tự
- Điểm Toán, Lý, Hóa : kiểu số thực

▪ Yêu cầu

- Lưu thông tin n SV?
- Truyền thông tin n SV vào hàm?

NTU - Kỹ thuật lập trình



1. Khái niệm kiểu cấu trúc

❖ Khai báo các biến để lưu trữ 1 SV

- char mssv[8]; // "52123456"
- char hoten[30]; // "Nguyen Van A"
- char ntts[8]; // "22/12/92"
- char phai; // 0
- float toan, ly, hoa; // 8.5 9.0 10.0

❖ Truyền thông tin 1 SV cho hàm

- void xuat(char *mssv, char *hoten, char *ntns, char phai, float toan, float ly, float hoa);

NTU - Kỹ thuật lập trình





1. Khái niệm kiểu cấu trúc

❖ Nhận xét

- Đặt tên biến khó khăn và khó quản lý
- Truyền tham số cho hàm quá nhiều
- Tìm kiếm, sắp xếp, sao chép,... khó khăn
- Tốn nhiều bộ nhớ
- ...

❖ Ý tưởng

- Gom những thông tin của cùng 1 SV thành một kiểu dữ liệu mới => Kiểu **struct**



2. Khai báo

❖ Khai báo cú pháp

```
struct <tên kiểu cấu trúc>
{
    <kiểu dữ liệu> <tên thành phần 1>;
    ...
    <kiểu dữ liệu> <tên thành phần n>;
};
```

❖ Ví dụ

```
struct DIEM
{
    int x;
    int y;
};
```





2. Khai báo

❖ Cú pháp tường minh

```
struct <tên kiểu cấu trúc>
{
    <kiểu dữ liệu> <tên thành phần 1>;
    ...
    <kiểu dữ liệu> <tên thành phần n>;
} <tên biến 1>, <tên biến 2>;
```

❖ Ví dụ

```
struct DIEM
{
    int x;
    int y;
} diem1, diem2;
```

NTU - Kỹ thuật lập trình



2. Khai báo

❖ Cú pháp không tường minh

```
struct <tên kiểu cấu trúc>
{
    <kiểu dữ liệu> <tên thành phần 1>;
    ...
    <kiểu dữ liệu> <tên thành phần n>;
};

struct <tên kiểu cấu trúc> <tên biến>;
```

❖ Ví dụ

```
struct DIEM
{
    int x;
    int y;
};

struct DIEM diem1, diem2; // C++ có thể bỏ struct
```

NTU - Kỹ thuật lập trình





2. Khai báo

❖ Cú pháp sử dụng typedef

```
typedef struct
{
    <kiểu dữ liệu> <tên thành phần 1>;
    ...
    <kiểu dữ liệu> <tên thành phần n>;
} <tên kiểu cấu trúc>;
<tên kiểu cấu trúc> <tên biến>;
```

❖ Ví dụ

```
typedef struct
{
    int x;
    int y;
} DIEM;
struct DIEM diem1, diem2;
```

NTU - Kỹ thuật lập trình



2. Khai báo

❖ Cú pháp tương minh

```
struct <tên kiểu cấu trúc>
{
    <kiểu dữ liệu> <tên thành phần 1>;
    ...
    <kiểu dữ liệu> <tên thành phần n>;
} <tên biến> = {<giá trị 1>, ..., <giá trị n>};
```

❖ Ví dụ

```
struct DIEM
{
    int x;
    int y;
} diem1 = {5, 8}, diem2;
```

NTU - Kỹ thuật lập trình





2. Khai báo

❖ Đặc điểm

- Không thể truy xuất trực tiếp
- Thông qua toán tử thành phần cấu trúc . hay còn gọi là **toán tử chấm** (dot operation)

<biến cấu trúc>. <tên thành phần>

❖ Ví dụ

```
struct DIEM
{
    int x;
    int y;
} diem1;
printf("x = %d, y = %d", diem1.x, diem1.y);
```

NTU - Kỹ thuật lập trình



2. Khai báo

❖ Có 2 cách

<biến cấu trúc đích> = <biến cấu trúc nguồn>;

<biến cấu trúc đích>. <tên thành phần> = <giá trị>;

❖ Ví dụ

```
struct DIEM
{
    int x, y;
} diem1 = {5, 8}, diem2;
...
diem2 = diem1;
diem2.x = diem1.x;
diem2.y = diem1.y * 2;
```

NTU - Kỹ thuật lập trình





3. Một số cấu trúc phức tạp

❖ Thành phần của cấu trúc là cấu trúc khác

```
struct DIEM
{
    int x;
    int y;
};

struct HINHCHUNHAT
{
    struct DIEM traitren;
    struct DIEM phaiduoai;
} hcnn1;
...
hcnn1.traitren.x = 2;
hcnn1.traitren.y = 5;
```

NTU - Kỹ thuật lập trình



3. Một số cấu trúc phức tạp

❖ Thành phần của cấu trúc là mảng

```
struct SINHVIEN
{
    char hoten[30];
    float toan, ly, hoa;
} sv1;
...
strcpy(sv1.hoten, "Nguyen Van A");
sv1.toan = 10;
sv1.ly = 6.5;
sv1.hoa = 9;
```

NTU - Kỹ thuật lập trình





3. Một số cấu trúc phức tạp

❖ Cấu trúc đệ quy (tự trỏ)

```
struct PERSON
{
    char hoten[30];
    struct PERSON *father, *mother;
};

struct NODE
{
    int value;
    struct NODE *pNext;
};
```

NTU - Kỹ thuật lập trình



3. Một số cấu trúc phức tạp

❖ Lưu ý

- **Kiểu** cấu trúc được định nghĩa **để làm khuôn dạng** còn **biến** cấu trúc được khai báo **để sử dụng** khuôn dạng **đã định nghĩa**.
- Trong C++, có thể bỏ từ khóa struct khi khai báo biến (hoặc sử dụng **typedef**)
- Khi **nhập các biến kiểu số thực** trong cấu trúc thường nhập thông qua một biến trung gian.

NTU - Kỹ thuật lập trình





3. Một số cấu trúc phức tạp

❖ Mảng cấu trúc

- Tương tự như mảng với KDL cơ sở (char, int, float, ...)

```
struct DIEM
{
    int x;
    int y;
};

DIEM mang1[20];
DIEM mang2[10] = {{3, 2}, {4, 4}, {2, 7}};
```



3. Một số cấu trúc phức tạp

❖ Truyền cấu trúc cho hàm

- Giống như truyền kiểu dữ liệu cơ sở
 - Tham trị (không thay đổi sau khi kết thúc hàm)
 - Tham chiếu
 - Con trả
- Ví dụ

```
struct DIEM { int x, y; };

void xuat1(int x, int y) { ... };
void xuat2(DIEM diem) { ... };
void xuat3(DIEM &diem) { ... };
void xuat4(DIEM *diem) { ... };
```





3. Một số cấu trúc phức tạp

❖ Truy xuất cấu trúc thông qua con trỏ

```
<đại lượng con trỏ cấu trúc> -> <đại lượng thành phần>
(*<đại lượng con trỏ cấu trúc>).<đại lượng thành phần>
```

❖ Ví dụ

```
typedef struct
{
    int tu, mau;
} PHANSO;
PHANSO ps1, *ps2 = &ps1; // ps2 là con trỏ

ps1.tu = 1; ps1.mau = 2;
ps2->tu = 1; ps2->mau = 2;
⇒ (*ps2).tu = 1; (*ps2).mau = 2;
```

NTU - Kỹ thuật lập trình



4. Kiểu union

❖ Khái niệm hợp nhất (union)

- Được khai báo và sử dụng như cấu trúc
- Các thành phần của union có chung địa chỉ đầu (nằm chồng lên nhau trong bộ nhớ)

❖ Khai báo

```
union <tên kiểu union>
{
    <kiểu dữ liệu> <tên thành phần 1>;
    ...
    <kiểu dữ liệu> <tên thành phần 2>;
};
```

NTU - Kỹ thuật lập trình





4. Kiểu union

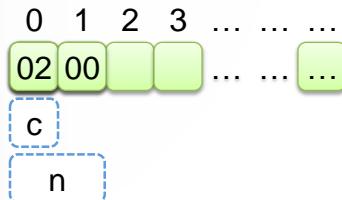
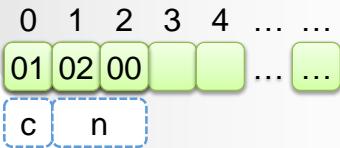
❖ So sánh struct và union

```
struct MYSTRUCT
{
    char c;
    int n;
} s;

s.c = 1; s.n = 2;
```

```
union MYUNION
{
    char c;
    int n;
} u;

u.c = 1; u.n = 2;
```



NTU - Kỹ thuật lập trình



5. Bài tập Chương 5

Viết chương trình thực hiện các chức năng:

- Nhập vào n sinh viên, mỗi sinh viên gồm các thông tin:
 - Họ lót
 - Tên
 - Điểm
 - Giới tính
- In ra danh sách sinh viên, mỗi người 1 dòng
- In danh sách các sinh viên đạt yêu cầu (Điểm \geq 5.0)
- Cho biết điểm trung bình của các sinh viên nam/nữ
- In danh sách sắp xếp theo vần A, B, C tên, họ
- In danh sách sắp xếp theo điểm giảm dần

NTU - Kỹ thuật lập trình





Chương 6

Xử lý tập tin

Bài giảng Kỹ thuật lập trình



Nội dung

1. Khái niệm cơ bản
2. Phân loại tập tin
3. Các thao tác trên tập tin
4. Một số hàm xử lý tập tin
5. Bài tập Chương 6



1. Khái niệm cơ bản

❖ Mở đầu

- C lưu trữ dữ liệu (biến, mảng, cấu trúc, ...) trong bộ nhớ RAM.
- Dữ liệu được nạp vào RAM và gửi ra ngoài thông qua các thiết bị (**device**)
 - **Thiết bị nhập** (input device): bàn phím, con chuột
 - **Thiết bị xuất** (output device): màn hình, máy in
 - **Thiết bị vừa nhập vừa xuất**: tập tin
- Các thiết bị đều thực hiện mọi xử lý thông qua các dòng (**stream**).



1. Khái niệm cơ bản

❖ Stream (dòng)

- Là môi trường trung gian để giao tiếp (nhận/ gửi thông tin) giữa chương trình và thiết bị.
- ➔ Muốn nhận/gửi thông tin cho một thiết bị ta sẽ gửi thông tin cho stream nối với thiết bị đó (**độc lập thiết bị**).
- **Stream** là dãy byte dữ liệu
 - “Chảy” vào chương trình gọi là **stream nhập**.
 - “Chảy” ra chương trình gọi là **stream xuất**.





1. Khái niệm cơ bản

❖ Phân loại Stream

- Stream **văn bản** (text)
 - Chỉ chứa các **ký tự**.
 - Tổ chức thành từng dòng, mỗi dòng tối đa 255 ký tự, kết thúc bởi ký tự cuối dòng '\0' hoặc ký tự sang dòng mới '\n'.
- Stream **nhi phân** (binary)
 - Chứa các **byte**.
 - Được đọc và ghi chính xác từng byte.
 - Xử lý dữ liệu bất kỳ, kể cả dữ liệu văn bản.
 - Được sử dụng chủ yếu với các tập tin trên đĩa.

NTU - Kỹ thuật lập trình



1. Khái niệm cơ bản

❖ Các stream chuẩn định nghĩa sẵn

Tên	Stream	Thiết bị tương ứng
stdin	Nhập chuẩn	Bàn phím
stdout	Xuất chuẩn	Màn hình
stderr	Lỗi chuẩn	Màn hình
stdprn (MS-DOS)	In chuẩn	Máy in (LPT1:)
stdaux (MS-DOS)	Phụ chuẩn	Cổng nối tiếp COM 1:

❖ Ví dụ (hàm **fprintf** xuất ra stream xác định)

- Xuất ra màn hình: `fprintf(stdout, "Hello");`
- Xuất ra máy in: `fprintf(stdprn, "Hello");`
- Xuất ra thiết bị báo lỗi: `fprintf(stderr, "Hello");`
- Xuất ra tập tin (stream **fp**): `fprintf(fp, "Hello");`

NTU - Kỹ thuật lập trình





1. Khái niệm cơ bản

❖ Nhu cầu sử dụng tập tin

- Dữ liệu giới hạn và được lưu trữ tạm thời
 - Nhập: gõ từ **bàn phím**.
 - Xuất: hiển thị trên **màn hình**.
 - Lưu trữ dữ liệu: trong **bộ nhớ RAM**.
- ➔ Mất thời gian, không giải quyết được bài toán với số dữ liệu lớn.
- Cần một thiết bị lưu trữ sao cho dữ liệu **vẫn còn khi kết thúc chương trình**, có thể **sử dụng nhiều lần** và **kích thước không hạn chế**.

NTU - Kỹ thuật lập trình



1. Khái niệm cơ bản

❖ Khái niệm tập tin

- **Tập hợp thông tin** (dữ liệu) được tổ chức theo một dạng nào đó với một tên xác định.
- Một **dãy byte liên tục** (ở góc độ lưu trữ).
- Được lưu trữ trong các thiết bị lưu trữ ngoài như đĩa mềm, đĩa cứng, USB...
 - **Vẫn tồn tại khi chương trình kết thúc.**
 - **Kích thước không hạn chế** (tùy vào thiết bị lưu trữ)
- Cho phép đọc dữ liệu (**thiết bị nhập**) và ghi dữ liệu (**thiết bị xuất**).

NTU - Kỹ thuật lập trình





1. Khái niệm cơ bản

❖ Quy tắc đặt tên tập tin

Tên
(name)

Mở rộng
(extension)

- Không bắt buộc.
- Thường có 3 ký tự.
- Thường do chương trình ứng dụng tạo tập tin tự đặt

- Bắt buộc phải có.
- Hệ điều hành MS-DOS: dài tối đa 8 ký tự.
- Hệ điều hành Windows: dài tối đa 128 ký tự.
- Gồm các ký tự A đến Z, số 0 đến 9, ký tự khác như #, \$, %, ~, ^, @, (,), !, _, khoảng trắng.

NTU - Kỹ thuật lập trình



1. Khái niệm cơ bản

❖ Đường dẫn

- Chỉ đến một tập tin không nằm trong thư mục hiện hành. Ví dụ: **c:\data\list.txt** chỉ tập tin **list.txt** nằm trong thư mục **data** của ổ đĩa C.
- Trong chương trình, đường dẫn này được ghi trong chuỗi như sau: "**c:\\data\\list.txt**"
- Dấu **** biểu thị ký tự điều khiển nên để thể hiện nó ta phải thêm một dấu **** ở trước. Nhưng nếu chương trình yêu cầu nhập đường dẫn từ bàn phím thì chỉ nhập một dấu ****.

NTU - Kỹ thuật lập trình





1. Khái niệm cơ bản

❖ Con trỏ chỉ vị (position indicator)

- Được tạo tự động khi mở tập tin.
- Xác định nơi diễn ra việc đọc/ghi trong tập tin

❖ Vị trí con trỏ chỉ vị

- Khi tập tin chưa mở: ở đầu tập tin (giá trị 0).
- Khi mở tập tin:
 - Ở cuối tập tin khi mở để chèn (mode **a** hay **a+**)
 - Ở đầu tập tin (hay giá trị 0) khi mở với các mode khác (**w**, **w+**, **r**, **r+**).



1. Khái niệm cơ bản

❖ Truy xuất tuần tự (sequentially access)

- Phải đọc/ghi dữ liệu từ vị trí con trỏ chỉ vị đến vị trí n-1 trước khi đọc dữ liệu tại vị trí n.
- **Không cần quan tâm đến con trỏ chỉ vị** do con trỏ chỉ vị tự động chuyển sang vị trí kế tiếp sau thao tác đọc/ghi dữ liệu.

❖ Truy xuất ngẫu nhiên (random access)

- Có thể đọc/ghi tại vị trí bất kỳ trong tập tin mà không cần phải đọc/ghi toàn bộ dữ liệu trước đó → **quan tâm đến con trỏ chỉ vị**.





2. Phân loại tập tin

❖ Các cách phân loại

- **Theo người sử dụng:** quan tâm đến nội dung tập tin nên sẽ phân loại theo phần mở rộng
➔ **.EXE, .COM, .CPP, .DOC, .PPT, ...**
- **Theo người lập trình:** tự tạo các stream tường minh để kết nối với tập tin xác định nên sẽ phân loại theo cách sử dụng stream trong C
➔ **tập tin kiểu văn bản** (ứng với stream văn bản) và **tập tin kiểu nhị phân** (ứng với stream nhị phân).



2. Phân loại tập tin

❖ Tập tin kiểu văn bản (stream văn bản)

- Dãy các dòng kế tiếp nhau.
- Mỗi dòng dài tối đa 255 ký tự và kết thúc bằng ký hiệu cuối dòng (**end_of_line**).
- **Dòng không phải là một chuỗi** vì không được kết thúc bởi ký tự '**\0**'.
- Khi ghi '**\n**' được chuyển thành cặp ký tự **CR** (về đầu dòng, mã ASCII 13) và **LF** (qua dòng, mã ASCII 10).
- Khi đọc thì cặp **CR-LF** được chuyển thành '**\n**'.





2. Phân loại tập tin

❖ Tập tin kiểu nhị phân (stream nhị phân)

- Dữ liệu được **đọc và ghi** một cách **chính xác**, không có sự **chuyển đổi**.
- Ký tự kết thúc chuỗi '\0' và **end_of_line** không có ý nghĩa là cuối chuỗi và cuối dòng mà **được xử lý** như mọi ký tự khác.



3. Các thao tác trên tập tin

❖ Quy trình thao tác với tập tin

- **Mở tập tin:** tạo một stream nối kết với tập tin cần mở, stream được quản lý bởi biến con trỏ đến cấu trúc **FILE**
 - Cấu trúc được định sẵn trong **STDIO.H**
 - Các thành phần của cấu trúc này được dùng trong các thao tác xử lý tập tin.
- **Sử dụng tập tin** (sau khi đã mở được tập tin)
 - **Đọc dữ liệu** từ tập tin đưa vào chương trình.
 - **Ghi dữ liệu** từ chương trình lên tập tin.
- **Đóng tập tin** (sau khi sử dụng xong).





3. Các thao tác trên tập tin

❖ **Đọc/ghi dữ liệu:**

- Nhập/xuất **theo định dạng**
 - Hàm: **fscanf, fprintf**
 - Chỉ dùng với tập tin **kiểu văn bản**.
- Nhập/xuất **từng ký tự hay dòng** lên tập tin
 - Hàm: **getc, fgetc, fgets, putc, fputs**
 - Chỉ nên dùng với **kiểu văn bản**.
- Đọc/ghi **trực tiếp** dữ liệu từ bộ nhớ lên tập tin
 - Hàm: **fread, fwrite**
 - Chỉ dùng với tập tin **kiểu nhị phân**.



4. Các hàm xử lý tập tin

❖ **Hàm nhập xuất tập tin (File I/O function)**

- Mở và đóng tập tin: **fopen, fclose**
- Nhập/Xuất tập tin:
 - Theo định dạng: **fprintf, fscanf**
 - Từng ký tự hay chuỗi: **fputc, fputs, fgetc, fgets**
 - Trực tiếp từ bộ nhớ: **fwrite, fread**

❖ **Hàm quản lý tập tin (File-Management function)**

- Xóa tập tin: **remove**
- Đổi tên tập tin: **rename**





4. Các hàm xử lý tập tin

❖ Hàm mở tập tin

FILE *fopen(const char *filename, const char *mode)



Mở tập tin có tên (đường dẫn) là chứa trong **filename** với kiểu mở **mode** (xem bảng).

- ◆ Thành công: con trỏ kiểu cấu trúc **FILE**
- ◆ Thất bại: **NULL** (sai quy tắc đặt tên tập tin, không tìm thấy ổ đĩa, không tìm thấy thư mục, mở tập tin chưa có để đọc, ...)

```
FILE* fp = fopen("taptin.txt", "rt");
if (fp == NULL)
    printf("Khong mo duoc tap tin!");
```

NTU - Kỹ thuật lập trình



4. Các hàm xử lý tập tin

Đối số

Ý nghĩa

b	Mở tập tin kiểu nhi phân (binary)
t	Mở tập tin kiểu văn bản (text) (mặc định)
r	Mở tập tin chỉ để đọc dữ liệu từ tập tin. Trả về NULL nếu không tìm thấy tập tin.
w	Mở tập tin chỉ để ghi dữ liệu vào tập tin. Tập tin sẽ được tạo nếu chưa có, ngược lại dữ liệu trước đó sẽ bị xóa hết.
a	Mở tập tin chỉ để thêm (append) dữ liệu vào cuối tập tin. Tập tin sẽ được tạo nếu chưa có.
r+	Giống mode r và bổ sung thêm tính năng ghi dữ liệu và tập tin sẽ được tạo nếu chưa có.
w+	Giống mode w và bổ sung thêm tính năng đọc.
a+	Giống mode a và bổ sung thêm tính năng đọc.

NTU - Kỹ thuật lập trình





4. Các hàm xử lý tập tin

❖ Hàm xuất theo định dạng

int fprintf(FILE *fp, char *fnt, ...)



Trả về



Ghi dữ liệu có chuỗi định dạng **fnt** (giống hàm printf) vào stream **fp**.

Nếu fp là **stdout** thì hàm giống printf.

- ◆ Thành công: trả về số byte ghi được.
- ◆ Thất bại: trả về **EOF** (có giá trị là -1, được định nghĩa trong STUDIO.H, sử dụng trong tập tin có kiểu văn bản)

```
int i = 2912; int c = 'P'; float f = 17.06;
FILE* fp = fopen("taptin.txt", "wt");
if (fp != NULL)
    fprintf(fp, "%d %c %.2f\n", i, c, f);
```



NTU - Kỹ thuật lập trình



4. Các hàm xử lý tập tin

❖ Hàm nhập theo định dạng

int fscanf(FILE *fp, char *fnt, ...)



Trả về



Đọc dữ liệu có chuỗi định dạng **fnt** (giống hàm scanf) từ stream **fp**.

Nếu fp là **stdin** thì hàm giống printf.

- ◆ Thành công: trả về số thành phần đọc và lưu trữ được.
- ◆ Thất bại: trả về **EOF**.

```
int i;
FILE* fp = fopen("taptin.txt", "rt");
if (fp != NULL)
    fscanf(fp, "%d", &i);
```



NTU - Kỹ thuật lập trình



4. Các hàm xử lý tập tin

❖ Ví dụ

- Một tập tin chứa nhiều dòng, mỗi dòng là thông tin mỗi sinh viên theo định dạng sau:
 - <MSV>-<Tên>(<Phái>)tab<NTNS>tab<ĐTB>
 - Ví dụ: 0312078-H. P. Trang(Nu) 17/06/85 8.5

❖ Đọc chuỗi thông tin phức hợp

- %[chuỗi]**: đọc cho đến khi **không gặp** ký tự nào trong chuỗi thì dừng.
- %[^chuỗi]**: đọc cho đến khi **gặp** một trong những ký tự trong chuỗi thì dừng.

NTU - Kỹ thuật lập trình



4. Các hàm xử lý tập tin

❖ Hàm nhập ký tự

int getc(FILE *fp) và int fgetc(FILE *fp)



Đọc một ký tự từ stream **fp**.
getc là macro còn **fgetc** là phiên bản hàm của macro **getc**.



- ◆ Thành công: trả về ký tự đọc được sau khi chuyển sang số nguyên không dấu.
- ◆ Thất bại: trả về **EOF** khi kết thúc stream **fp** hoặc gặp lỗi.



```
char ch;
FILE* fp = fopen("taptin.txt", "rt");
if (fp != NULL)
    ch = getc(fp); // ⇔ ch = fgetc(fp);
```

NTU - Kỹ thuật lập trình





4. Các hàm xử lý tập tin

❖ Hàm nhập chuỗi ký tự

int fgets(char *str, int n, FILE *fp)



Đọc một dãy ký tự từ stream **fp** vào vùng nhớ **str**,
kết thúc khi đủ **n-1** ký tự hoặc gặp ký tự xuống
dòng.

Trả về

- ◆ Thành công: trả về **str**.
- ◆ Thất bại: trả về **NULL** khi gặp lỗi hoặc gặp ký tự **EOF**.



```
char s[20];
FILE* fp = fopen("taptin.txt", "rt");
if (fp != NULL)
    fgets(s, 20, fp);
```

KT
LT
269

NTU - Kỹ thuật lập trình



4. Các hàm xử lý tập tin

❖ Hàm xuất ký tự

int putc(int ch, FILE *fp)/int fputc(in ch, FILE *fp)



Ghi ký tự **ch** vào stream **fp**.
putc là macro còn **fputc** là phiên bản hàm của
macro **putc**.

Trả về

- ◆ Thành công: trả về ký tự **ch**.
- ◆ Thất bại: trả về **EOF**.



```
FILE* fp = fopen("taptin.txt", "rt");
if (fp != NULL)
    putc('a', fp); // hoặc fputc('a', fp);
```

KT
LT
270

NTU - Kỹ thuật lập trình



4. Các hàm xử lý tập tin

❖ Hàm xuất chuỗi ký tự

int fputs(const char *str, FILE *fp)



Ghi chuỗi ký tự **str** vào stream **fp**. Nếu **fp** là **stdout** thì **fputs** giống hàm **puts**, nhưng **puts** ghi ký tự xuống dòng.



- ◆ Thành công: trả về ký tự cuối cùng đã ghi.
- ◆ Thất bại: trả về **EOF**.



```
char s[] = "Ky thuat lap trinh";
FILE* fp = fopen("taptin.txt", "wt");
if (fp != NULL)
    fputs(s, fp);
```

KT
LT
271

NTU - Kỹ thuật lập trình



4. Các hàm xử lý tập tin

❖ Hàm xuất trực tiếp

int fwrite(void *buf, int size, int count, FILE *fp)



Ghi **count** mẫu tin có kích thước mỗi mẫu tin là **size** (byte) từ vùng nhớ **buf** vào stream **fp** (theo kiểu nhị phân).



- ◆ Thành công: trả về số lượng mẫu tin (không phải số lượng byte) đã ghi.
- ◆ Thất bại: số lượng nhỏ hơn **count**.



```
int a[] = {1, 2, 3};
FILE* fp = fopen("taptin.dat", "wb");
if (fp != NULL)
    fwrite(a, sizeof(int), 3, fp);
```

KT
LT
272

NTU - Kỹ thuật lập trình



4. Các hàm xử lý tập tin

❖ Hàm nhập trực tiếp

int fread(void *buf, int size, int count, FILE *fp)



Đọc **count** mẫu tin có kích thước mỗi mẫu tin là **size** (byte) vào vùng nhớ **buf** từ stream **fp** (theo kiểu nhị phân).

- ◆ Thành công: trả về số lượng mẫu tin (không phải số lượng byte) thật sự đã đọc.
- ◆ Thất bại: số lượng nhỏ hơn **count** khi kết thúc stream **fp** hoặc gặp lỗi.

```
int a[5];
FILE* fp = fopen("taptin.dat", "wb");
if (fp != NULL)
    fread(a, sizeof(int), 3, fp);
```

KT
LT
273

NTU - Kỹ thuật lập trình



4. Các hàm xử lý tập tin

❖ Hàm đóng tập tin

int fclose(FILE *fp)



Đóng stream **fp**.

Dữ liệu trong stream **fp** sẽ được “vét” (ghi hết lên đĩa) trước khi đóng.

- ◆ Thành công: trả về 0.
- ◆ Thất bại: trả về **EOF**.

```
FILE* fp = fopen("taptin.txt", "rt");
...
fclose(fp);
```

KT
LT
274

NTU - Kỹ thuật lập trình



4. Các hàm xử lý tập tin

❖ Hàm đóng tất cả stream

int fcloseall()



Đóng tất cả stream đang được mở ngoại trừ các stream chuẩn `stdin`, `stdout`, `stdprn`, `stderr`, `stdaux`. Nên đóng từng stream thay vì đóng tất cả.



- ◆ Thành công: trả về số lượng stream được đóng.
- ◆ Thất bại: trả về `EOF`.



```
FILE* fp1 = fopen("taptin1.txt", "rt");
FILE* fp2 = fopen("taptin2.txt", "wt");
...
fcloseall();
```



NTU - Kỹ thuật lập trình



4. Các hàm xử lý tập tin

❖ “Vét” dữ liệu trong stream

- Khi chương trình kết thúc, các stream đang mở sẽ được “vét” (`flush`) và đóng lại. Tuy nhiên, ta nên đóng một cách tường minh các stream sau khi sử dụng xong (nhất là các stream tập tin) để tránh các sự cố xảy ra trước khi chương trình kết thúc bình thường.
- Ta có thể “vét” dữ liệu trong stream mà không cần đóng stream đó bằng một trong hai hàm:
 - Vét stream `fp` xác định: `int fflush(FILE *fp);`
 - Vét tất cả stream đang mở: `int flushall();`



NTU - Kỹ thuật lập trình



4. Các hàm xử lý tập tin

❖ Hàm đặt lại vị trí con trỏ chỉ vị

void rewind(FILE *fp)



Đặt lại vị trí con trỏ chỉ vị về đầu (byte 0) tập tin fp.



♦ Không



```
FILE* fp = fopen("taptin.txt", "w+");
fprintf(fp, "0123456789");
rewind(fp);
fprintf(fp, "*****");
```



LT

277

NTU - Kỹ thuật lập trình



4. Các hàm xử lý tập tin

❖ Hàm tái định vị con trỏ chỉ vị

int fseek(FILE *fp, long offset, int origin)



Đặt vị trí con trỏ chỉ vị trong stream fp với vị trí offset so với cột mốc origin (SEEK_SET hay 0: đầu tập tin; SEEK_CUR hay 1: vị trí hiện tại; SEEK_END hay 2: cuối tập tin)



- ♦ Thành công: trả về 0.
- ♦ Thất bại: trả về giá trị khác 0.



```
FILE* fp = fopen("taptin.txt", "w+");
fseek(fp, 0L, SEEK_SET); // ⇔ rewind(fp);
fseek(fp, 0L, SEEK_END); // cuối tập tin
fseek(fp, -2L, SEEK_CUR); // lùi lại 2 vị trí
```



LT

278

NTU - Kỹ thuật lập trình



4. Các hàm xử lý tập tin

❖ Hàm xác định vị trí con trỏ chỉ vị

long ftell(FILE *fp)



Hàm trả về vị trí hiện tại của con trỏ chỉ vị (tính từ vị trí đầu tiên của tập tin, tức là 0) của stream **fp**.



- ◆ Thành công: trả về vị trí hiện tại của con trỏ chỉ vị.
- ◆ Thất bại: trả về **-1L**.



```
FILE* fp = fopen("taptin.txt", "rb");
fseek(fp, 0L, SEEK_END);
long size = ftell(fp);
printf("Kich thuoc tap tin la %ld\n", size);
```

NTU - Kỹ thuật lập trình

KT
279



4. Các hàm xử lý tập tin

❖ Dấu hiệu kết thúc tập tin

- Khi đã biết kích thước tập tin
 - Sử dụng fwrite để lưu n mẫu tin
➔ kích thước = n * sizeof(1 mẫu tin);
 - Sử dụng hàm fseek kết hợp hàm ftell
- Khi chưa biết kích thước tập tin
 - Hằng số **EOF** (= -1) (**chỉ cho tập tin văn bản**)
 - ➔ while ((c = fgetc(fp)) != EOF) ...
 - Hàm int **feof**(FILE *fp) (cho cả 2 kiểu tập tin)
 - ➔ trả về số 0 nếu chưa đến cuối tập tin
 - ➔ trả về số khác 0 nếu đã đến cuối tập tin.



4. Các hàm xử lý tập tin

❖ Hàm xóa tập tin

int remove(const char *filename)



Xóa tập tin xác định bởi **filename**.



- ◆ Thành công: trả về 0.
- ◆ Thất bại: trả về -1.



```
if (remove("c:\\vc.txt") == 0)
    printf("Tap tin vc.txt da bi xoa!");
else
    printf("Ko xoa duoc tap tin vc.txt!");
```

KT
LT
281

NTU - Kỹ thuật lập trình



4. Các hàm xử lý tập tin

❖ Hàm đổi tên tập tin

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



Đổi tên tập tin **oldname** thành **newname**.

Hai tập tin **phải cùng ổ đĩa** nhưng không cần thiết **phải cùng thư mục** (có thể sử dụng để di chuyển hay sao chép tập tin).



- ◆ Thành công: trả về 0.
- ◆ Thất bại: trả về -1.



```
if (rename("c:\\a.txt", "c:\\BT\\b.cpp") == 0)
    printf("Doi ten tap tin thanh cong");
else
    printf("Doi ten tap tin that bai");
```

KT
LT
282

NTU - Kỹ thuật lập trình



5. Bài tập Chương 6

1. Phân biệt giữa stream kiểu văn bản và kiểu nhị phân?
2. Khi mở tập tin bằng fopen, ta cần phải xác định thông tin nào và hàm sẽ trả về cái gì?
3. Ba phương pháp để truy xuất tập tin?
4. Hai phương pháp để đọc thông tin từ tập tin là gì?
5. Con trỏ chỉ vị là gì và cách thay đổi nó?
6. Giá trị và mục đích dùng hằng ký hiệu EOF?
7. Quy trình xử lý tập tin?
8. Cách xác định cuối tập tin kiểu văn bản và nhị phân?
9. Nếu mở một tập tin chưa có (bằng mode w), cho biết giá trị của con trỏ chỉ vị lúc đầu?

NTU - Kỹ thuật lập trình



Chương 7

Lập trình

đệ quy

Bài giảng Kỹ thuật lập trình

Nội dung

- 1. Khái niệm cơ bản
- 4. Một số dạng hàm đặc biệt đê quy
- 3. Các vấn đề thông dụng
- 4. Kỹ thuật khử đệ quy
- 5. Bài tập Chương 7

NTU - Kỹ thuật lập trình



1. Khái niệm cơ bản

❖ Mở đầu

- Trong nhiều tình huống việc mô tả các bài toán sẽ đơn giản và hiệu quả hơn nếu ta nhìn được nó dưới góc độ mang tính đệ qui.
- Mô tả mang tính đệ qui về một đối tượng là mô tả theo cách phân tích đối tượng thành nhiều thành phần mà trong số đó có thành phần mang tính chất của chính đối tượng được mô tả. Tức là mô tả đối tượng qua chính nó.



❖ 2 điều kiện quan trọng

- Tồn tại bước đệ quy.
- Điều kiện dừng.

NTU - Kỹ thuật lập trình





1. Khái niệm cơ bản

❖ Ví dụ

- Cho $S(n) = 1 + 2 + 3 + \dots + n$
=> $S(10)$? $S(11)$?

$$S(10) = 1 + 2 + \dots + 10 = 55$$

$$S(11) = 1 + 2 + \dots + 10 + 11 = 66$$

$$= S(10) + 11$$

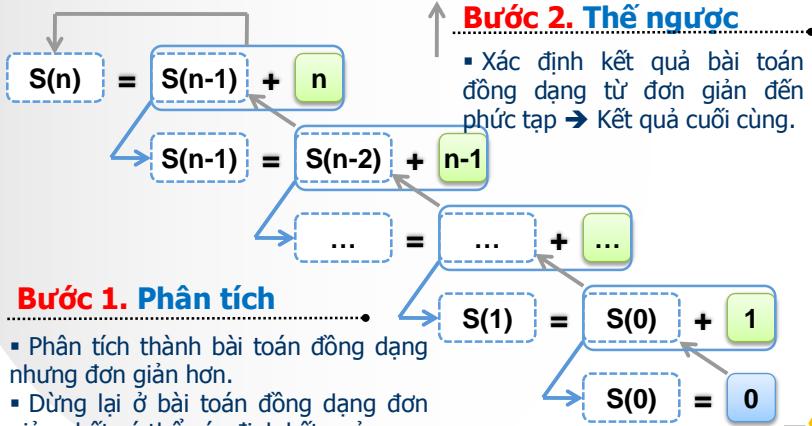
$$= 55 + 11 = 66$$

NTU - Kỹ thuật lập trình



1. Khái niệm cơ bản

❖ 2 bước giải bài toán



NTU - Kỹ thuật lập trình

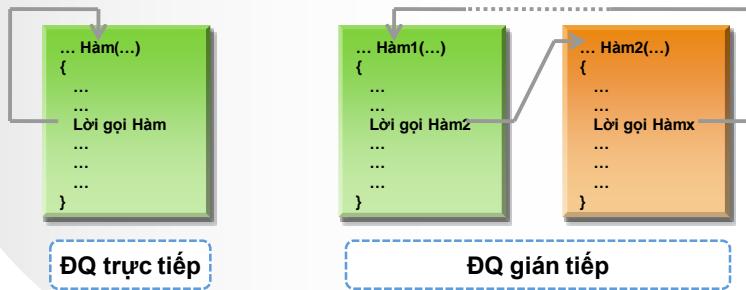




1. Khái niệm cơ bản

❖ Khái niệm hàm đệ quy

- Một hàm được gọi là đệ quy nếu bên trong thân của hàm đó có lời gọi hàm lại chính nó một cách trực tiếp hay gián tiếp.

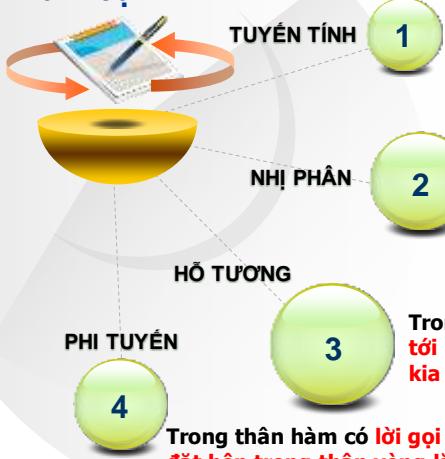


NTU - Kỹ thuật lập trình



1. Khái niệm cơ bản

❖ Phân loại



Trong thân hàm có **duy nhất** một lời gọi hàm gọi lại chính nó một cách tường minh.

Trong thân hàm có **hai lời** gọi hàm gọi lại chính nó một cách tường minh.

Trong thân hàm này có lời gọi hàm tới hàm kia và bên trong thân hàm kia có lời gọi hàm tới hàm này.

Trong thân hàm có **lời gọi hàm** lại chính nó **được đặt** bên trong **thân vòng lặp**.

NTU - Kỹ thuật lập trình





1. Khái niệm cơ bản

▪ Đệ quy tuyến tính



Cấu trúc chương trình

```
<Kiểu> TênHàm(<TS>) {
    if (<ĐK dừng>)
        ...
        return <Giá Trị>;
    ...
    TênHàm(<TS>);
}
```

NTU - Kỹ thuật lập trình

Ví dụ

Tính $S(n) = 1 + 2 + \dots + n$
 $\Rightarrow S(n) = S(n-1) + n$
ĐK dừng: $S(0) = 0$

∴ Chương trình ∴
long Tong(int n)
{
 if (n == 0)
 return 0;
 return **Tong(n-1) + n;**
}



1. Khái niệm cơ bản

▪ Đệ quy nhị phân



Cấu trúc chương trình

```
<Kiểu> TênHàm(<TS>) {
    if (<ĐK dừng>)
        ...
        return <Giá Trị>;
    ...
    TênHàm(<TS>);
    ...
    TênHàm(<TS>);
}
```

NTU - Kỹ thuật lập trình

Ví dụ

Tính số hạng thứ n của dãy
Fibonacy:
 $f(0) = f(1) = 1$
 $f(n) = f(n-1) + f(n-2)$ $n > 1$
ĐK dừng: $f(0) = 1$ và $f(1) = 1$

∴ Chương trình ∴
long Fibo(int n)
{
 if (n == 0 || n == 1)
 return 1;
 return **Fibo(n-1)+Fibo(n-2);**
}





1. Khái niệm cơ bản

▪ Đệ quy hỗn hợp



Cấu trúc chương trình

```
<Kiểu> TênHàm1(<TS>) {
    if (<ĐK dừng>)
        return <Giá trị>;
    ... TênHàm2(<TS>); ...
}
<Kiểu> TênHàm2(<TS>) {
    if (<ĐK dừng>)
        return <Giá trị>;
    ... TênHàm1(<TS>); ...
}
```

NTU - Kỹ thuật lập trình

Ví dụ

Tính số hạng thứ n của dãy:

$$\begin{aligned}x(0) &= 1, y(0) = 0 \\x(n) &= x(n-1) + y(n-1) \\y(n) &= 3*x(n-1) + 2*y(n-1)\end{aligned}$$

ĐK dừng: $x(0) = 1, y(0) = 0$

∴ Chương trình ∴

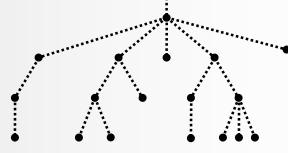
```
long yn(int n);
long xn(int n) {
    if (n == 0) return 1;
    return xn(n-1)+yn(n-1);
}
long yn(int n) {
    if (n == 0) return 0;
    return 3*xn(n-1)+2*yn(n-1);
}
```

KT
LT
293



1. Khái niệm cơ bản

▪ Đệ quy phi tuyến



Cấu trúc chương trình

```
<Kiểu> TênHàm(<TS>) {
    if (<ĐK dừng>)
        ...
        return <Giá Trị>;
    }
    ... Vòng lặp {
        ... TênHàm(<TS>); ...
    }
}
```

NTU - Kỹ thuật lập trình

Ví dụ

Tính số hạng thứ n của dãy:

$$\begin{aligned}x(0) &= 1 \\x(n) &= n^2x(0) + (n-1)^2x(1) + \\&\dots + 2^2x(n-2) + 1^2x(n-1)\end{aligned}$$

ĐK dừng: $x(0) = 1$

∴ Chương trình ∴

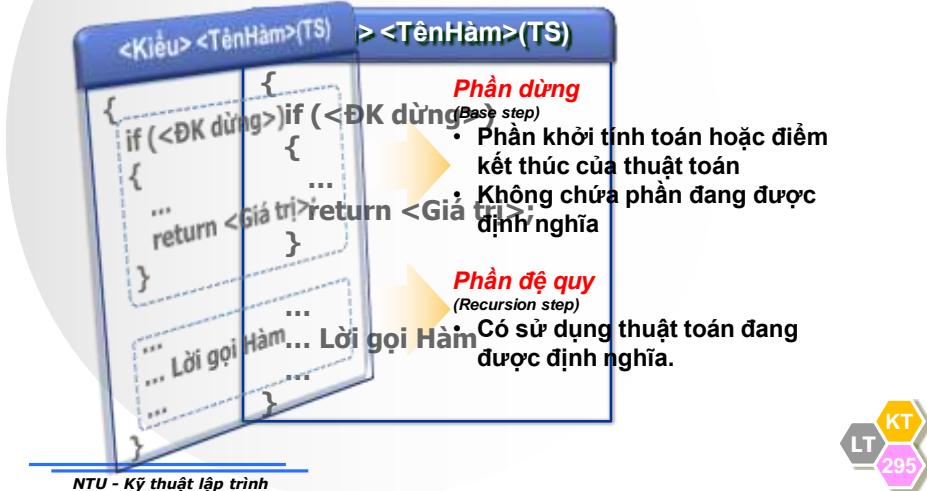
```
long xn(int n)
{
    if (n == 0) return 1;
    long s = 0;
    for (int i=1; i<=n; i++)
        s = s + i*i*xn(n-i);
    return s;
}
```

KT
LT
294



2. Phân tích hàm đệ quy

❖ Cấu trúc hàm đệ quy



2. Phân tích hàm đệ quy

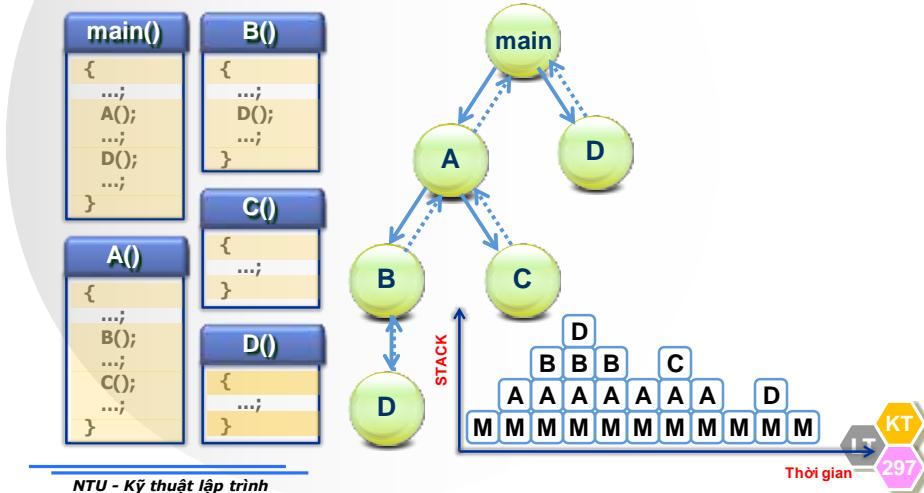
❖ Các bước xây dựng hàm đệ quy





2. Phân tích hàm đệ quy

❖ Cơ chế gọi hàm và STACK



2. Phân tích hàm đệ quy

❖ Cơ chế gọi hàm dùng STACK trong C phù hợp cho giải thuật đệ quy vì:

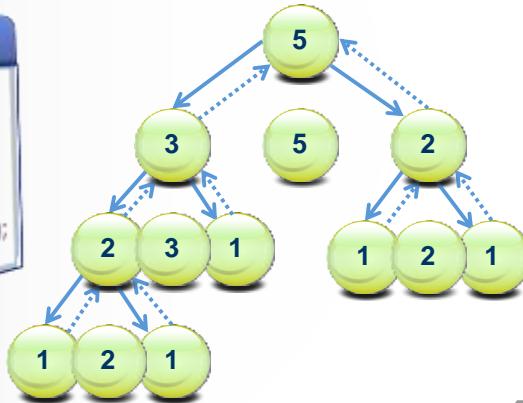
- Lưu thông tin trạng thái còn dang mở mỗi khi gọi đệ quy.
- Thực hiện xong một lần gọi cần khôi phục thông tin trạng thái trước khi gọi.
- Lệnh gọi cuối cùng sẽ hoàn tất đầu tiên.



2. Phân tích hàm đệ quy

- ❖ Ví dụ: Tính số hạng thứ 4 của dãy Fibonacy

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



NTU - Kỹ thuật lập trình



2. Phân tích hàm đệ quy

- ❖ Phân tích giải thuật đệ quy

- Sử dụng cây đệ quy (recursive tree)
 - Giúp hình dung bước phân tích và thế ngược.
 - Bước phân tích: đi từ trên xuống dưới.
 - Bước thế ngược đi từ trái sang phải, từ dưới lên trên.
 - Ý nghĩa
 - Chiều cao của cây \Leftrightarrow Độ lớn trong STACK.
 - Số nút \Leftrightarrow Số lời gọi hàm.

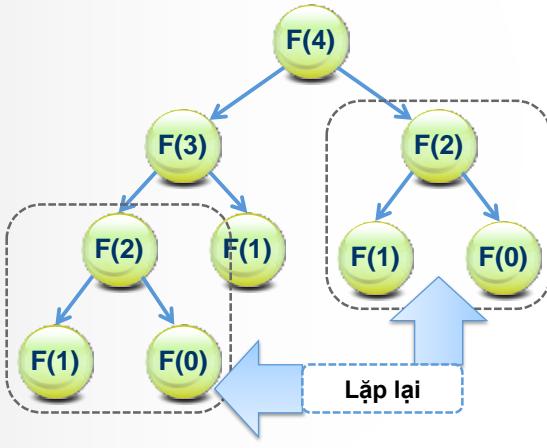
NTU - Kỹ thuật lập trình





2. Phân tích hàm đệ quy

❖ Ví dụ: cây đệ quy Fibonacy



NTU - Kỹ thuật lập trình



2. Phân tích hàm đệ quy

- Ví dụ: tính tổng n số lẻ đan dẫu đầu tiên

$$S_n = 1 - 3 + 5 - 7 \dots + (-1)^{n+1} * (2n-1)$$

```
int Sn(int n)
{
    if n == 1
        return 1;
    else
        if (n%2)
            return Sn(n-1) - (2*n-1);
        else
            return Sn(n-1) + (2*n-1);
}
```

NTU - Kỹ thuật lập trình





2. Phân tích hàm đệ quy

- **Ví dụ:** tìm USCLN của 2 số nguyên (cách 1)

```
int USCLN(int m, int n)
{
    if (n == 0)
        return m;
    else
        return USCLN(n, m % n);
}
```



2. Phân tích hàm đệ quy

- **Ví dụ:** tìm USCLN của 2 số nguyên (cách 2)

```
int USCLN(int m, int n)
{
    if(m == 0)
        return n;
    else
        if(m>n)
            return USCLN(m-n, n);
        else
            return USCLN(n-m, m);
}
```





2. Phân tích hàm đệ quy

- **Ví dụ:** đổi số cơ số 10 sang cơ số $2 \leq m \leq 9$

```
int A[max]; i=0;
void Convert(int n, int i)
{
    if (n != 0)
    {
        A[i] = n % m ;
        Convert(n/m, i+1);
    }
}
```



2. Phân tích hàm đệ quy

❖ Một số lỗi thường gặp

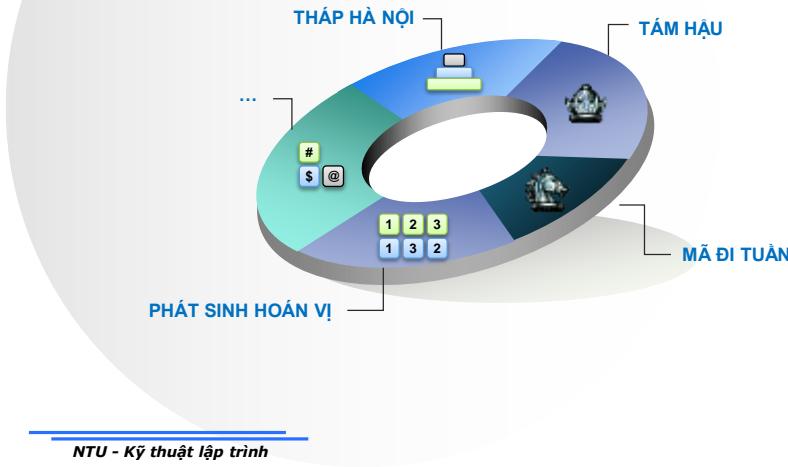
- Công thức đệ quy chưa đúng, không tìm được bài toán đồng dạng đơn giản hơn (không hội tụ) nên không giải quyết được vấn đề.
- Không xác định các trường hợp suy biến – neo (điều kiện dừng).
- Thông điệp thường gặp là **StackOverflow** do:
 - Thuật giải đệ quy đúng nhưng số lần gọi đệ quy quá lớn làm tràn STACK.
 - Thuật giải đệ quy sai do không hội tụ hoặc không có điều kiện dừng.





2. Phân tích hàm đệ quy

❖ Một số bài toán kinh điển



2. Phân tích hàm đệ quy

□ Tháp Hà Nội

❖ Mô tả bài toán

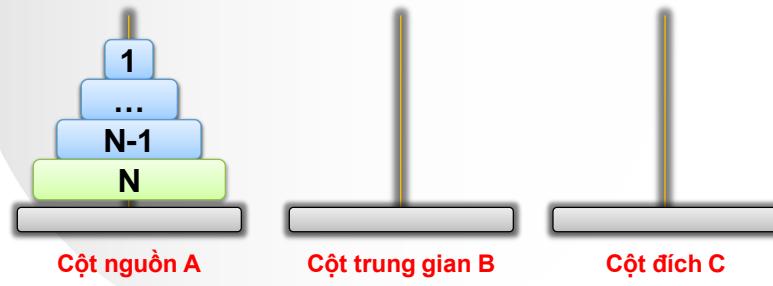
- Có 3 cột A, B và C và cột A hiện có N đĩa.
- Tìm cách chuyển N đĩa từ cột A sang cột C sao cho:
 - Một lần chuyển 1 đĩa
 - Đĩa lớn hơn phải nằm dưới.
 - Có thể sử dụng các cột A, B, C làm cột trung gian.





2. Phân tích hàm đệ quy

$$N \text{ đĩa } A \rightarrow C = ? \text{ đĩa } A \rightarrow B + \text{ Đĩa } N A \rightarrow C + N-1 \text{ đĩa } B \rightarrow C$$



2. Phân tích hàm đệ quy

❖ Giải thuật tổng quát

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





2. Phân tích hàm đệ quy

```

void hanoi(int from, int to, int dia)
{
    int trunggian;
    if (dia == 1)
        printf("\nChuyen 1 dia tu coc %c sang coc %c", 'A'+from, 'A'+to);
    else
    {
        if ((from == A && to == C) || (from == C && to == A))
            trunggian = B;
        else if ((from == A && to == B) || (from == B && to == A))
            trunggian = C;
        else if ((from == C && to == B) || (from == B && to == C))
            trunggian = A;
        hanoi(from, trunggian, dia-1);
        hanoi(from, to, 1);
        hanoi(trunggian, to, dia-1);
    }
}

```

NTU - Kỹ thuật lập trình

KT

311



2. Phân tích hàm đệ quy

□ Tám hậu

❖ Mô tả bài toán

- Cho bàn cờ vua kích thước 8x8
- Hãy đặt 8 hoàng hậu lên bàn cờ này sao cho không có hoàng hậu nào “ăn” nhau:
 - Không nằm trên cùng dòng, cùng cột
 - Không nằm trên cùng đường chéo xuôi, ngược.

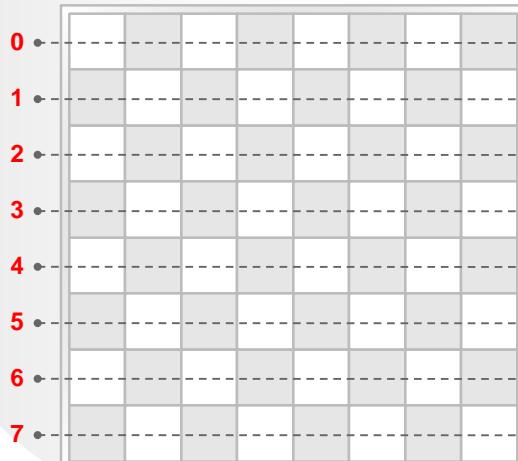
NTU - Kỹ thuật lập trình

KT
312
LT



2. Phân tích hàm đệ quy

❖ Hàng

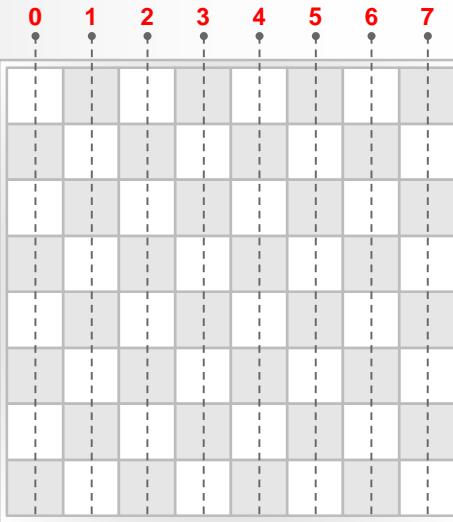


NTU - Kỹ thuật lập trình



2. Phân tích hàm đệ quy

❖ Cột



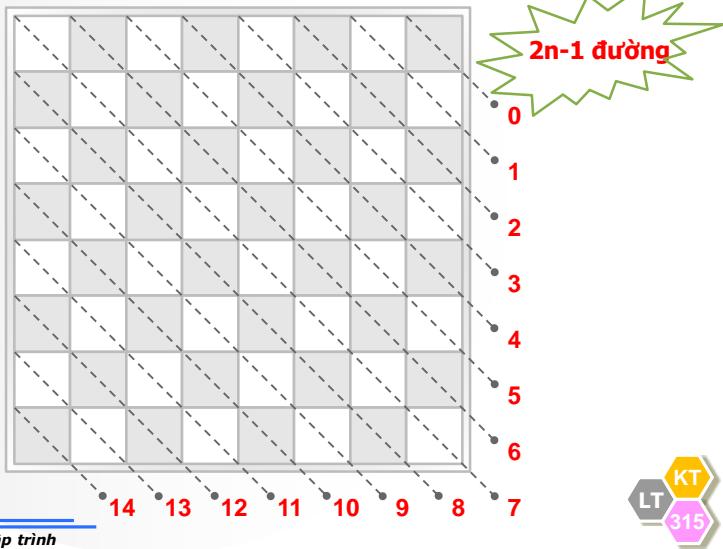
Lập trình
NTU - Kỹ thuật lập trình





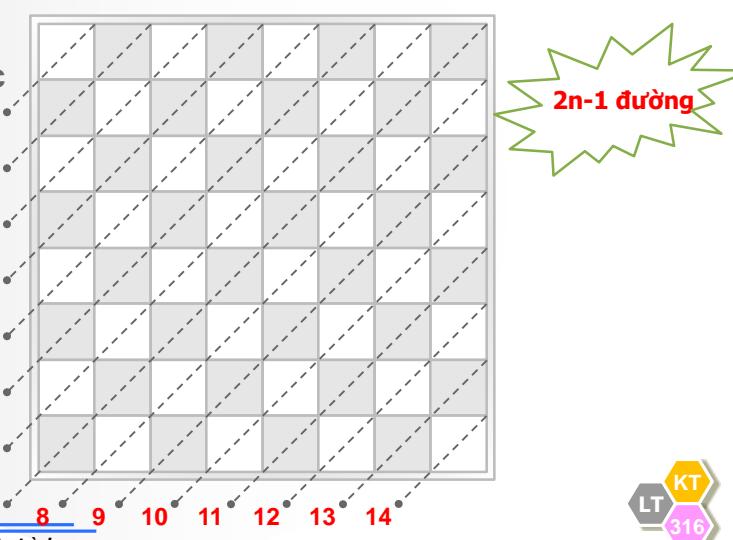
2. Phân tích hàm đệ quy

❖ Chéo
xuôi



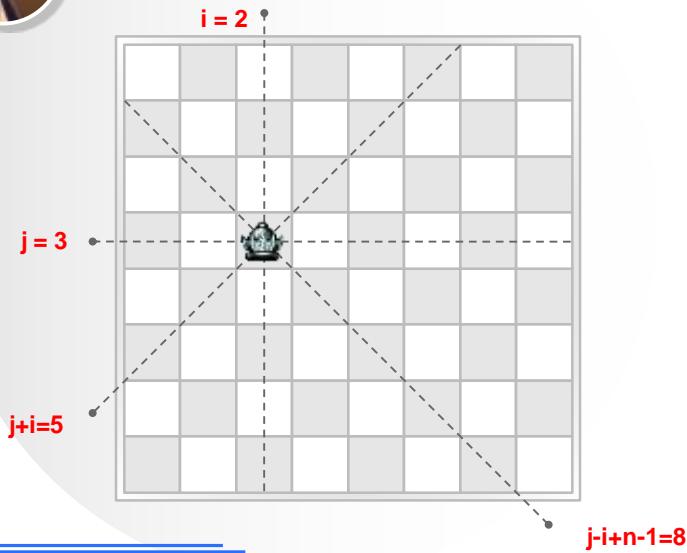
2. Phân tích hàm đệ quy

❖ Chéo
ngược





2. Phân tích hàm đệ quy



NTU - Kỹ thuật lập trình



2. Phân tích hàm đệ quy

```
void thu(int i)
{
    int j;
    for (j=0; j<8; j++)
    {
        if (cot[j] == 1 && cheoxuoi[i+j] == 1 && cheonguoc[i-j+7] == 1)
        {
            dong[i] = j; cot[j] = 0;
            cheoxuoi[i+j] = 0; cheonguoc[i-j+7] = 0;
            if (i<7)
                thu(i+1);
            else
                print();
            cot[j] = 1; cheoxuoi[i+j] = 1; cheonguoc[i-j+7] = 1;
        }
    }
}
```

NTU - Kỹ thuật lập trình





2. Phân tích hàm đệ quy

```
void tim()
{
    int i, q;
    for (i=0; i<8; i++)
    {
        cot[i] = 1;
        dong[i] = -1;
    }
    for (i=0; i<15; i++)
    {
        cheoxuoi[i] = 1;
        cheonguoc[i] = 1;
    }
    thu(0);
}
```

NTU - Kỹ thuật lập trình

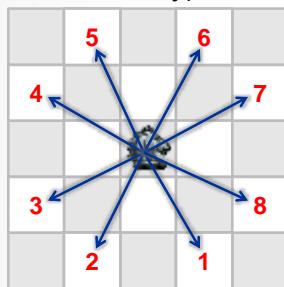


2. Phân tích hàm đệ quy

Mã đi tuần

❖ Mô tả bài toán

- Cho bàn cờ vua kích thước 8x8 (64 ô)
- Hãy đi con mã 64 nước sao cho mỗi ô chỉ đi qua 1 lần (xuất phát từ ô bất kỳ) theo luật:

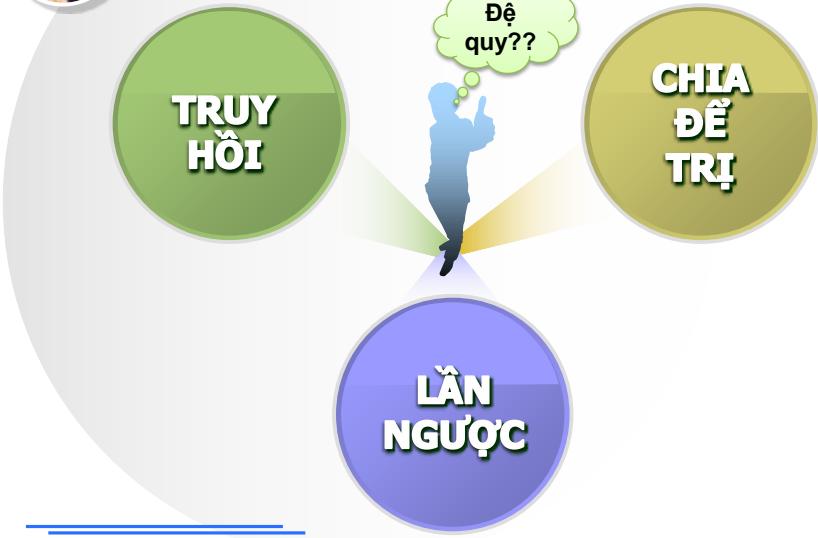


NTU - Kỹ thuật lập trình





3. Các vấn đề đệ quy thông dụng



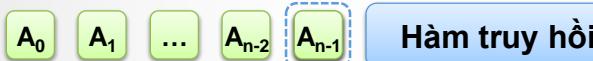
NTU - Kỹ thuật lập trình



3.1. Hệ thức truy hồi

❖ Khái niệm

- Hệ thức truy hồi của 1 dãy A là công thức biểu diễn phần tử A thông qua 1 hoặc nhiều số hạng trước của dãy.



NTU - Kỹ thuật lập trình





3.1. Hệ thức truy hồi

❖ **Ví dụ 1**

- Vi trùng cứ 1 giờ lại nhân đôi. Vậy sau 5 giờ sẽ có mấy con vi trùng nếu ban đầu có 2 con?

❖ **Giải pháp**

- Gọi V_h là số vi trùng tại thời điểm h .
- Ta có:
 - $V_h = 2V_{h-1}$
 - $V_0 = 2$

➔ Đệ quy tuyển tính với $V(h)=2^*V(h-1)$ và điều kiện dừng $V(0) = 2$



3.1. Hệ thức truy hồi

❖ **Ví dụ 2**

- Gửi ngân hàng 1000 USD, lãi suất 12%/năm. Số tiền có được sau 30 năm là bao nhiêu?

❖ **Giải pháp**

- Gọi T_n là số tiền có được sau n năm.
- Ta có:
 - $T_n = T_{n-1} + 0.12T_{n-1} = 1.12T_{n-1}$
 - $V(0) = 1000$

➔ Đệ quy tuyển tính với $T(n)=1.12*T(n-1)$ và điều kiện dừng $V(0) = 1000$

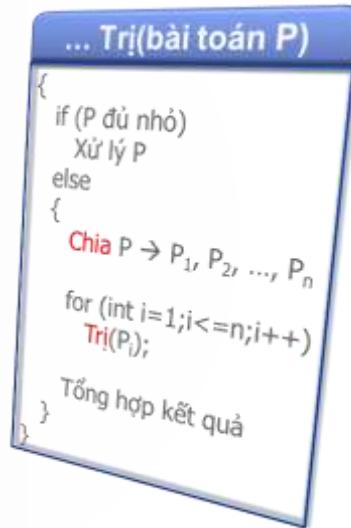




3.2.Chia để trị (divide & conquer)

❖ Khái niệm

- Chia bài toán thành nhiều bài toán con.
- Giải quyết từng bài toán con.
- Tổng hợp kết quả từng bài toán con để ra lời giải.



NTU - Kỹ thuật lập trình



3.2.Chia để trị (divide & conquer)

❖ Ví dụ 1

- Cho dãy A đã sắp xếp thứ tự tăng. Tìm vị trí phần tử x trong dãy (nếu có)

❖ Giải pháp

- $mid = (l + r) / 2;$
- Nếu $A[mid] = x \rightarrow$ trả về mid.
- Ngược lại
 - Nếu $x < A[mid] \rightarrow$ tìm trong đoạn $[l, mid - 1]$
 - Ngược lại \rightarrow tìm trong đoạn $[mid + 1, r]$

➔ Sử dụng đệ quy nhị phân.



NTU - Kỹ thuật lập trình



3.2.Chia để trị (divide & conquer)

❖ Ví dụ 2

- Tính tích 2 chuỗi số cực lớn X và Y

❖ Giải pháp

- $X = X_{2n-1} \dots X_n X_{n-1} \dots X_0, Y = Y_{2n-1} \dots Y_n Y_{n-1} \dots Y_0$
- Đặt $X_L = X_{2n-1} \dots X_n, X_N = X_{n-1} \dots X_0 \Rightarrow X = 10^n X_L + X_N$
- Đặt $Y_L = Y_{2n-1} \dots Y_n, Y_N = Y_{n-1} \dots Y_0 \Rightarrow Y = 10^n Y_L + Y_N$
- ➔ $X * Y = 10^{2n} X_L Y_L + 10^n (X_L Y_L + X_N Y_N) + X_N Y_N$
và $X_L Y_L + X_N Y_N = (X_L - X_N)(Y_N - Y_L) + X_L Y_L + X_N Y_N$
- ➔ Nhân 3 số nhỏ hơn (độ dài $\frac{1}{2}$) đến khi có thể nhân được ngay.



3.2.Chia để trị (divide & conquer)

❖ Một số bài toán khác

- Bài toán tháp Hà Nội
- Các giải thuật sắp xếp: QuickSort, MergeSort
- Các giải thuật tìm kiếm trên cây nhị phân tìm kiếm, cây nhị phân nhiều nhánh tìm kiếm.

❖ Lưu ý

- Khi bài toán lớn được chia thành các bài toán nhỏ hơn mà những bài toán nhỏ hơn này không đơn giản nhiều so với bài toán gốc thì **không nên** dùng kỹ thuật chia để trị.





3.3. Lần ngược (Backtracking)

❖ Khái niệm

- Tại bước có nhiều lựa chọn, ta chọn thử 1 bước để đi tiếp.
- Nếu không thành công thì “lần ngược” chọn bước khác.
- Nếu đã thành công thì ghi nhận lời giải này đồng thời “lần ngược” để truy tìm lời giải mới.
- Thích hợp giải các bài toán kinh điển như bài toán 8 hậu và bài toán mã đi tuần.

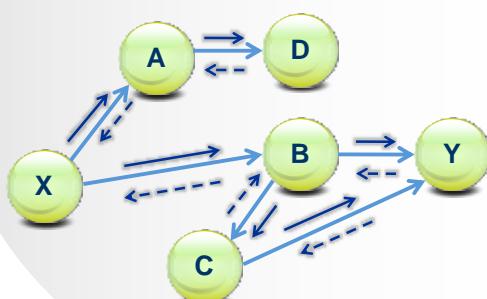
NTU - Kỹ thuật lập trình



3.3. Lần ngược (Backtracking)

❖ Ví dụ

- Tìm đường đi từ X đến Y.



NTU - Kỹ thuật lập trình





4. Khử đệ quy

❖ Nhận xét về giải thuật đệ quy

- **Ưu điểm**
 - Sáng sửa, dễ hiểu, nêu rõ bản chất vấn đề.
 - Tiết kiệm thời gian thực hiện mã nguồn.
 - Một số bài toán rất khó giải nếu không dùng đệ qui.
- **Khuyết điểm**
 - Tốn nhiều bộ nhớ, thời gian thực thi lâu.
 - Một số tính toán có thể bị lặp lại nhiều lần.
 - Một số bài toán không có lời giải đệ quy.



4. Khử đệ quy

❖ Đánh giá

- Chỉ nên dùng phương pháp đệ quy để giải các bài toán kinh điển như giải các vấn đề “chia để trị”, “lần ngược”.
- Vấn đề đệ quy không nhất thiết phải giải bằng phương pháp đệ quy, có thể sử dụng phương pháp khác thay thế (**khử đệ quy**)
- Tiện cho người lập trình nhưng không tối ưu khi chạy trên máy.
- Bước đầu nên giải bằng đệ quy nhưng từng bước khử đệ quy để nâng cao hiệu quả.





4. Khử đệ quy

❖ Khái niệm về khử đệ quy

- Thông thường khi gặp một bài toán khó giải quyết theo hướng không đệ quy ta thực hiện quá trình như sau:
 - Dùng quan niệm đệ quy để tìm giải thuật cho bài toán
 - Mã hoá giải thuật đệ quy
 - Khử đệ quy để có một chương trình không đệ quy.
- Quá trình trên gọi là khử đệ quy
- Trong một số trường hợp, việc khử đệ quy cũng không dễ dàng, nên phải chấp nhận chương trình đệ quy!



4. Khử đệ quy

❖ Hàm tính giá trị của dãy dữ liệu mô tả bằng hồi quy

- **Ví dụ 1:** hàm tính giai thừa không đệ quy

```
long int GiaiThua(int n)
{
    long int F =1;
    for (int i = 1; i <= n; i++)
        F = F * i;
    return F;
}
```





4. Khử đệ quy

- **Ví dụ 2:** hàm tính số Fibonacci lớn nhất bé hơn n

```
int Fibo (int n)
{
    int fib1 = 1, fib2 = 1, fib = 2;
    while (fib1+fib2<n)
    {
        fib = fib1 + fib2;
        fib2 = fib1;
        fib1 = fib;
    }
    return fib;
}
```



4. Khử đệ quy

- **Ví dụ 3:** tính tổng n số lẻ đan dẫu đầu tiên

$$S_n = 1 - 3 + 5 - 7 \dots + (-1)^{n+1} * (2n-1)$$

```
int Sn(int n)
{
    int k = 1 , tg = 1 ;
    while (k < n)
    {
        k++;
        if (k%2) tg += 2 * k - 1;
        else      tg -= 2 * k + 1;
    }
    return tg;
}
```





4. Khử đệ quy

❖ Thủ tục có dạng đệ quy đuôi

- **Ví dụ 1:** đổi số cơ số 10 sang cơ số $2 \leq m \leq 9$

```
void Convert(int n, int m)
{
    int A[max];
    While (n != 0)
    {
        A[m] = n % k;
        n = n / k;
        m = m - 1;
    }
}
```

NTU - Kỹ thuật lập trình



4. Khử đệ quy

- **Ví dụ 2:** tìm USCLN của 2 số nguyên (cách 1)

```
int USCLN(int m, int n)
{
    while(n != 0)
    {
        int sodu = m % n;
        m = n;
        n = sodu;
    }
    return m;
}
```

NTU - Kỹ thuật lập trình





4. Khử đệ quy

- **Ví dụ 3:** tìm USCLN của 2 số nguyên (cách 2)

```
int USCLN(int m, int n)
{
    while(n != 0)
    {
        int t1 = m;
        int t2 = n;
        m = abs(t1-t2);
        if(t1<t2) n = t1;
        else n = t2;
    }
    return m;
}
```

NTU - Kỹ thuật lập trình



5. Bài tập Chương 5

1. Viết hàm đệ quy xác định chiều dài chuỗi.

2. Hiển thị n dòng của tam giác Pascal.

- $a[i][0] = a[i][i] = 1$
- $a[i][k] = a[i-1][k-1] + a[i-1][k]$

Dòng 0: 1

Dòng 1: 1 1

Dòng 2: 1 2 1

Dòng 3: 1 3 3 1

Dòng 4: 1 4 6 4 1

...

NTU - Kỹ thuật lập trình





5. Bài tập Chương 5

3. Viết hàm đệ quy tính $C(n, k)$ biết
 - $C(n, k) = 1$ nếu $k = 0$ hoặc $k = n$
 - $C(n, k) = 0$ nếu $k > n$
 - $C(n, k) = C(n-1, k) + C(n-1, k-1)$ nếu $0 < k < n$
4. Cài đặt hoàn chỉnh Bài toán đổi 1 số thập phân sang cơ số khác.
5. Cài đặt hoàn chỉnh Bài toán “Tháp Hà Nội”.
6. Cài đặt hoàn chỉnh Bài toán “8 hậu”.
7. Cài đặt hoàn chỉnh Bài toán “Mã đi tuần”.

NTU - Kỹ thuật lập trình



**Xin trân trọng cảm ơn
Thầy Đặng Bình Phương và
các Thầy Cô – những tác giả của
các tài liệu mà tôi đã tham khảo và
sử dụng để hoàn thành bài giảng này!**

**Bùi Đức Dương
Khoa CNTT – Trường ĐH Nha Trang**