

Kỹ thuật lập trình

Programming Techniques

Ts. Nguyễn Đức Thuận
BM Hệ thống Thông Tin



Giới thiệu môn học

- **Đối tượng:** Sinh viên chuyên ngành Công nghệ thông tin
- **Môn học tiên quyết:** Nhập môn lập trình
- **Thời lượng:** 3TC (LT:2 + TH:1)

Tài liệu tham khảo:

1. Slide bài giảng: GV. Đặng Bình Phương, KTLT
2. Bài tập lập trình C nâng cao – Nguyễn Hữu Ngự
3. Một số vấn đề chọn lọc trong tin học – Ng Xuân My, Hồ Sĩ Đàm, Trần Đỗ Hùng, Lê Sĩ Quang
4. Cấu trúc dữ liệu & Giải Thuật - Lê Minh Hoàng
5. Data structure and Algorithms – A.V.Aho.J. Ullman, J.E. Hopcroft

Giới thiệu môn học

Tài liệu tham khảo:

6. Tài liệu giáo khoa Chuyên Tin (Quyển 1,2,3) Hồ Sĩ Đàm (chủ biên)
7. Một số vấn đề đáng chú ý trong Tin học, Phan Công Minh (chủ biên)
8. <https://www.geeksforgeeks.org/>
9. Slide bài giảng: GV Trịnh Thành Trung – BKHN
10. Tài liệu môn học: Download với

Email: ktlt.ntu@gmail.com PW: ktlt59th



Giới thiệu môn học

Nội dung môn học

- Chương 1: Kỹ thuật tổ chức chương trình
- Chương 2:
- Chương 3:
- Chương 4:
- Chương 5:

Ng Đức Thuận

Chương 1: KỸ THUẬT TỔ CHỨC CHƯƠNG TRÌNH

I. Tổng quan về kỹ thuật lập trình

“ Kỹ thuật lập trình là kỹ thuật thực thi một giải pháp phần mềm (cấu trúc dữ liệu + giải thuật) dựa trên nền tảng một phương pháp luận (methodology) và một hoặc nhiều ngôn ngữ lập trình phù hợp với yêu cầu đặc thù của ứng dụng”

▪ Kỹ thuật lập trình =

Tư tưởng thiết kế + Kỹ thuật mã hóa

Cấu trúc dữ liệu + Giải thuật + Ngôn ngữ lập trình

▪ **Lập trình** (Programming)

➤ Với mỗi bài toán (vấn đề) đặt ra, cần:

- ❖ *Thiết kế giải thuật để giải quyết bài toán đó*
- ❖ *Cài đặt giải thuật bằng một chương trình máy tính*

Chương 1: KỸ THUẬT TỔ CHỨC CHƯƠNG TRÌNH

I. Tổng quan về kỹ thuật lập trình

Thế nào là lập trình tốt?

❖ Đúng / Chính xác

- ✓ Thỏa mãn các nhiệm vụ
- ✓ Được khách hàng chấp nhận

❖ Ổn định

- ✓ Ổn định
- ✓ Ít lỗi hoặc lỗi nhẹ có thể chấp nhận được

❖ Khả năng nâng cấp

- ✓ Dễ dàng chỉnh sửa
- ✓ Dễ dàng nâng cấp trong điều kiện bài toán thay đổi

❖ Tái sử dụng

- ✓ Tái sử dụng hoặc kế thừa cho bài toán khác

Chương 1: KỸ THUẬT TỔ CHỨC CHƯƠNG TRÌNH

I. Tổng quan về kỹ thuật lập trình

Thế nào là lập trình tốt?

❖ Hiệu quả

- ✓ Thời gian lập trình ngắn
- ✓ Khả năng bảo trì dễ dàng
- ✓ Giá trị sử dụng lại lớn
- ✓ Sử dụng đơn giản, thân thiện
- ✓ Nhiều chức năng tiện ích

❖ Tương thích

- ✓ Thích ứng tốt các môi trường khác nhau

❖ Hiệu suất

- ✓ Chương trình nhỏ gọn, ít bộ nhớ
- ✓ Tốc độ nhanh, sử dụng ít CPU

Chương 1: KỸ THUẬT TỔ CHỨC CHƯƠNG TRÌNH

I. Tổng quan về kỹ thuật lập trình

Làm thế nào để lập trình tốt?

- ❖ Tư duy và phương pháp lập trình
- ❖ Hiểu sâu về máy tính
- ❖ Nắm vững ngôn ngữ
- ❖ Rèn luyện

Chương 1: KỸ THUẬT TỔ CHỨC CHƯƠNG TRÌNH

I. Tổng quan về kỹ thuật lập trình

A. Chương trình máy tính & Ngôn ngữ lập trình

(*Computer Program & Programming Language*)

- Chương trình máy tính

Tập hợp các lệnh chỉ dẫn cho máy tính thực hiện nhiệm vụ

- Ngôn ngữ lập trình

là 1 hệ thống các ký hiệu dùng để liên lạc, trao đổi 1 nhiệm vụ/ thuật toán với máy tính, làm cho nhiệm vụ được thực thi.

Có khoảng 1000 ngôn ngữ (60's đã có hơn 700) – phần lớn là các ngôn ngữ hàn lâm, có mục đích riêng hay phát triển bởi 1 tổ chức để phục vụ cho bản thân họ.

Chương 1: KỸ THUẬT TỔ CHỨC CHƯƠNG TRÌNH

B. Các thành phần cơ bản của 1 ngôn ngữ lập trình

- *Mô thức ngôn ngữ - Language paradigm*: nguyên tắc chung cơ bản của NNLT
- *Cú pháp – Syntax*: Cấu trúc ngữ pháp của ngôn ngữ lập trình
- *Ngữ nghĩa – Semantics*: Tác dụng, ý nghĩa của câu lệnh

- Về cơ bản, chỉ có 4 mô thức chính:

- ❖ *Imperative (Procedural) Paradigm* (Fortran, Pascal, C, Ada,)

- ❖ *Object-Oriented Paradigm* (SmallTalk, Java, C++)

- ❖ *Logic Paradigm* (Prolog)

- ❖ *Functional Paradigm* (Lisp, ML, Haskell)

- *Một vài Mô thức mới khác*:

- Concurrent programming,
- Distributed programming,

Chương 1: KỸ THUẬT TỔ CHỨC CHƯƠNG TRÌNH

C. Những tính chất cần có với các chương trình phần mềm:

- Tính mềm dẻo/mở rộng **scalability** / Khả năng chỉnh sửa **modifiability**
- Khả năng tích hợp **integrability** / Khả năng tái sử dụng **reusability**
- Tính chuyển đổi, linh hoạt, độc lập phần cứng - **portability**
- Hiệu năng cao - **performance**
- Độ tin cậy - **reliability**
- Dễ xây dựng
- Rõ ràng, dễ hiểu
- Ngắn gọn, xúc tích

Chương 1: KỸ THUẬT TỔ CHỨC CHƯƠNG TRÌNH

D. Hoạt động của 1 chương trình:

❖ Chương trình máy tính (Computer program) được nạp vào bộ nhớ *chính*



Trình dịch

⇒ 1 tập các lệnh bằng ngôn ngữ máy, tức là một dãy tuần tự các số nhị phân - *binary digits*



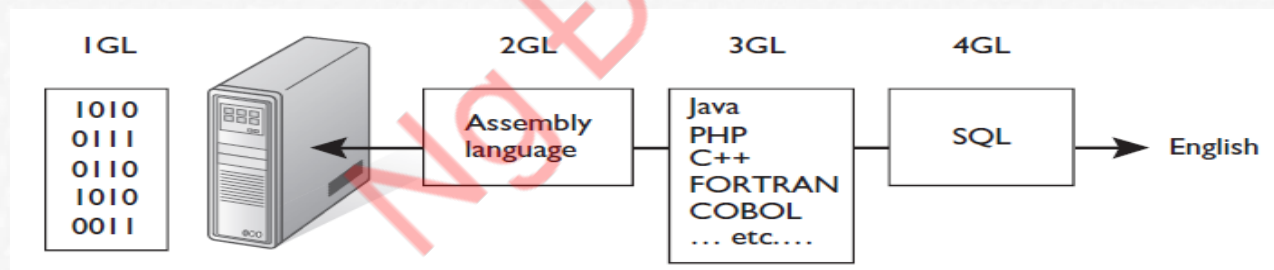
Các câu lệnh lần lượt được nhặt ra thi hành nhờ con trỏ lệnh - *instruction pointer*

❖ Thứ tự thực hiện các nhóm lệnh mã máy được gọi là luồng điều khiển *flow of control*

Chương 1: KỸ THUẬT TỔ CHỨC CHƯƠNG TRÌNH

E. Các thế hệ của ngôn ngữ lập trình (Language Generations)

Thế hệ	Phân loại
1st	Machine languages
2nd	Assembly languages
3rd	Procedural languages (High-level Language)
4th	Application languages (NonProcedural Lang.)
5th	Knowledge-based? Natural Language



Tham khảo thêm

<https://www.techopedia.com/definition/24308/fourth-generation-programming-language-4gl>

Chương 1: KỸ THUẬT TỔ CHỨC CHƯƠNG TRÌNH

Lịch sử phát triển

- ❖ 1940s : Machine code
- ❖ 1950s Khai thác sức mạnh của MT: Assembler code, Autocodes, first version of Fortran
- ❖ 1960s Tăng khả năng tính toán: Cobol, Lisp, Algol 60, Basic, PL/1 --- nhưng vẫn dùng phong cách lập trình cơ bản của assembly language.
- ❖ 1970s Ngôn ngữ có cấu trúc Ví dụ : Pascal, Algol 68 and C.
- ❖ 1980s Giảm sự phức tạp – object orientation, functional programming.
- ❖ 1990s Khai thác phần cứng song song và phân tán (parallel và distributed). Ví dụ NNLT chuyên parallel occam
- ❖ 2000s Genetic programming languages, DNA computing, bio-computing?
- ❖ Trong tương lai : Ngôn ngữ LT lượng tử : Quantum ?

Chương 1: KỸ THUẬT TỔ CHỨC CHƯƠNG TRÌNH

F. Phân loại ngôn ngữ

Có thể phân loại ngôn ngữ theo các tiêu chí khác nhau

❖ **Ngôn ngữ cấp thấp & ngôn ngữ cấp cao** (Low & High level Language)

Mức	Câu lệnh	Xử lý bộ nhớ
Cấp thấp	Dạng bits	Truy cập và cấp phát trực tiếp bộ nhớ
Cấp cao	Dùng các biểu thức và các dòng điều khiển	Truy cập và cấp phát bộ nhớ qua các lệnh, toán tử
Cấp rất cao	Hoàn toàn trừu tượng, độc lập phần cứng	Che dấu hoàn toàn việc truy cập và tự động cấp phát bộ

Chương 1: KỸ THUẬT TỔ CHỨC CHƯƠNG TRÌNH

F. Phân loại ngôn ngữ

Có thể phân loại ngôn ngữ theo các tiêu chí khác nhau

❖ Ngôn ngữ tường thuật & không tường thuật

(**Declarative & imperative** language)

❖Nhóm 1 gọi là **Declarative** (tường thuật - chính là functional và logic languages)

Ví dụ: SQL, XQuery.

❖Nhóm 2 gọi là **imperative** hay procedural (tức là các ngôn ngữ thủ tục, mệnh lệnh).

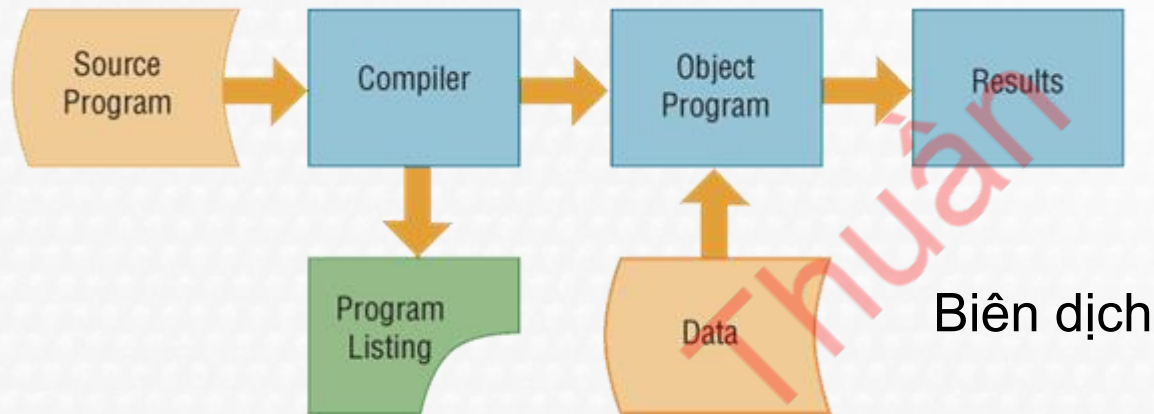
G. Chuyển đổi chương trình (trình dịch)

Có 2 loại trình dịch là: Biên dịch (Compiler) và thông dịch (interpreter)

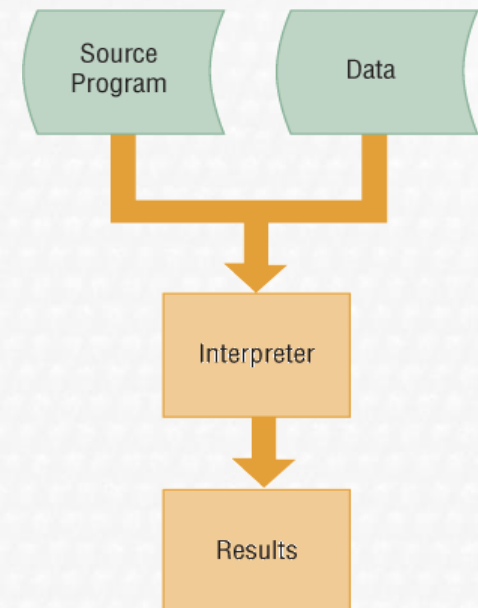
Biên dịch: Là chương trình thực hiện biên dịch toàn bộ các lệnh của chương trình nguồn thành mã máy trước khi thực hiện

Thông dịch: Là chương trình dịch và thực hiện từng dòng lệnh của chương trình.

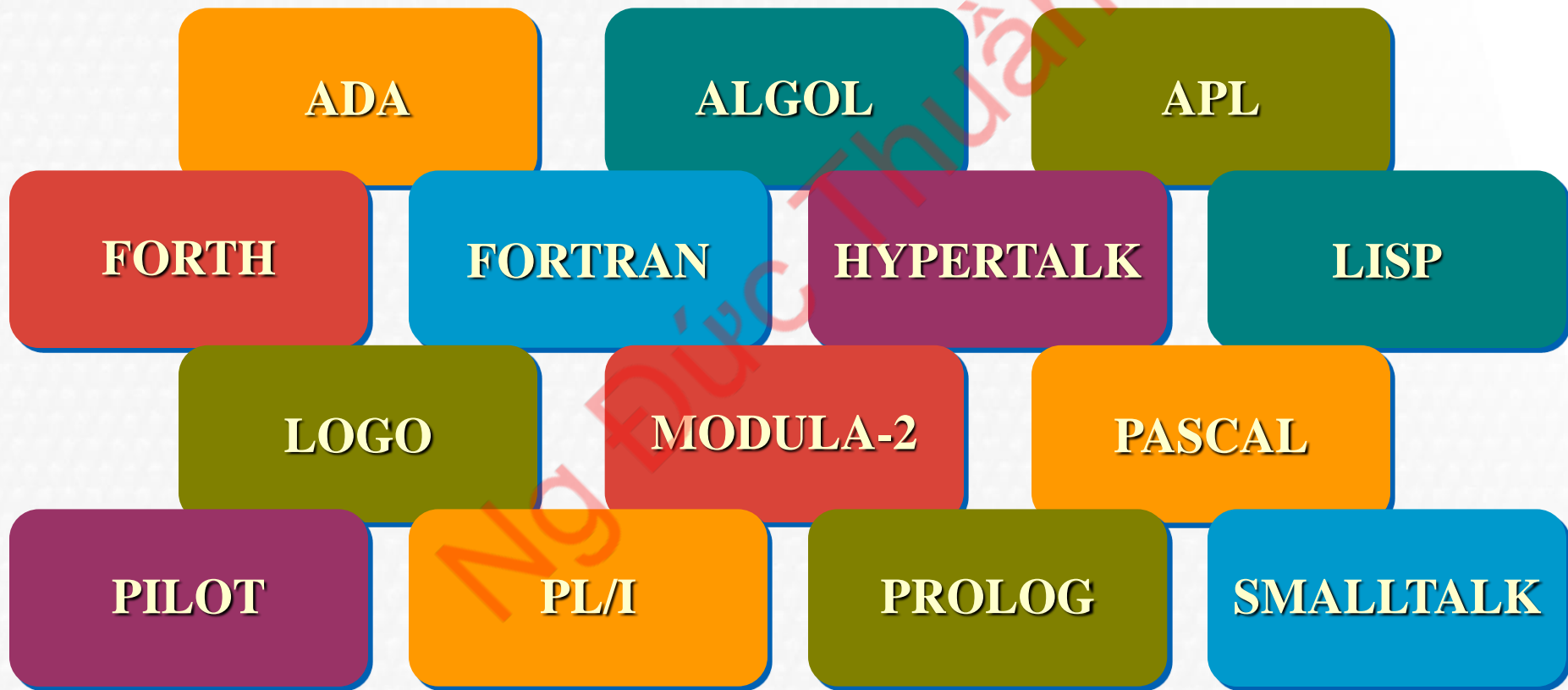
Chương 1: KỸ THUẬT TỔ CHỨC CHƯƠNG TRÌNH



Thông dịch



Một số ngôn ngữ lập trình



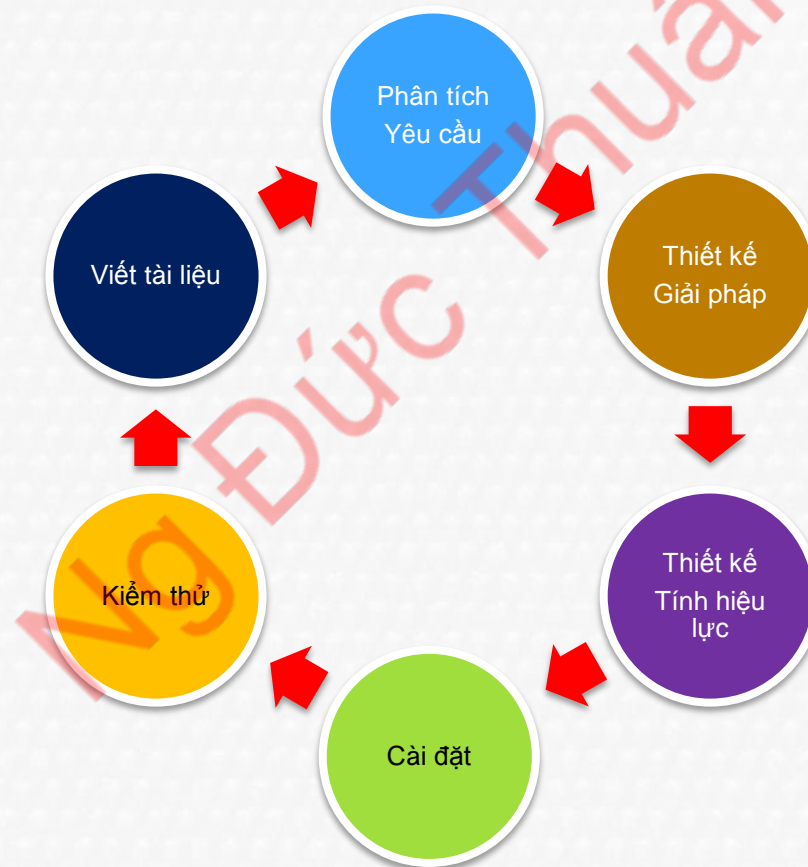
Mười ngôn ngữ lập trình đáng quan tâm 2019

- Python
- Java
- C/C++
- JavaScript
- Go programming language
- R
- Swift
- PHP
- C#
- MATLAB

Ng Đức Thuận

Chương 1: KỸ THUẬT TỔ CHỨC CHƯƠNG TRÌNH

■ II. Tổ chức mã nguồn cho đề án phần mềm



Chương 1: KỸ THUẬT TỔ CHỨC CHƯƠNG TRÌNH

■ II. Tổ chức mã nguồn cho đề án phần mềm

Các bước xây dựng mã nguồn (chương trình) cho 1 đề án phần mềm

Bước 1: Phân tích yêu cầu (*Analyze Requirements*)

Các bước thực hiện:

1. **Khảo sát và Thiết lập các yêu cầu**
2. **XD các mô hình phân tích**
3. **Xác định đầu vào, đầu ra và các xử lý cùng các thành phần dữ liệu.**

Chương 1: KỸ THUẬT TỔ CHỨC CHƯƠNG TRÌNH

■ II. Tổ chức mã nguồn cho đề án phần mềm

Các bước xây dựng mã nguồn (chương trình) cho 1 đề án phần mềm

Bước 2: Thiết kế giải pháp (*Design Solution*)

Phân chia hệ thống từng bước
thành các thủ tục để giải quyết

```
graph TD; A[Phân chia hệ thống từng bước thành các thủ tục để giải quyết] --> B[Thiết kế hướng đối tượng]; A --> C[Thiết kế hướng cấu trúc (phân rã chức năng)];
```

Thiết kế hướng đối
tượng

Thiết kế hướng cấu trúc
(phân rã chức năng)

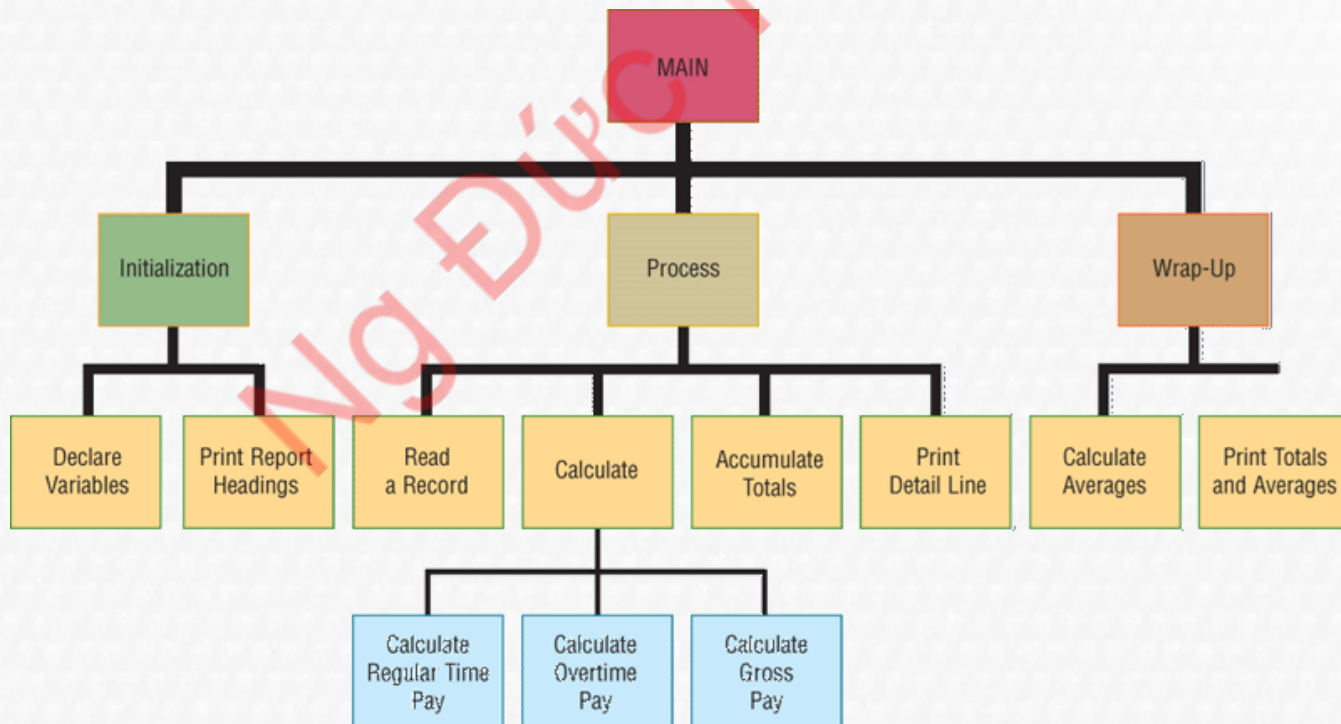
Chương 1: KỸ THUẬT TỔ CHỨC CHƯƠNG TRÌNH

■ II. Tổ chức mã nguồn cho đề án phần mềm

Các bước xây dựng mã nguồn (chương trình) cho 1 đề án phần mềm

Bước 2: Thiết kế giải pháp (Design Solution)

Xây dựng sơ đồ phân rã chức năng (*hierarchical chart*): Trực quan hóa các thủ tục, chức năng



Chương 1: KỸ THUẬT TỔ CHỨC CHƯƠNG TRÌNH

■ II. Tổ chức mã nguồn cho đề án phần mềm

Các bước xây dựng mã nguồn (chương trình) cho 1 đề án phần mềm

Bước 2: Thiết kế giải pháp (*Design Solution*)

Thiết kế hướng đối tượng:

➤ đóng gói dữ liệu và các thủ tục xử lý dữ liệu trong 1 **object**

- Các objects được nhóm lại thành các **classes**
- Biểu đồ lớp thể hiện trực quan các quan hệ phân cấp: quan hệ của các **classes**

Chương 1: KỸ THUẬT TỔ CHỨC CHƯƠNG TRÌNH

■ II. Tổ chức mã nguồn cho đề án phần mềm

Các bước xây dựng mã nguồn (chương trình) cho 1 đề án phần mềm

Bước 3: Xác lập tính hợp lệ (Validation Design)



Chương 1: KỸ THUẬT TỔ CHỨC CHƯƠNG TRÌNH

■ II. Tổ chức mã nguồn cho đề án phần mềm

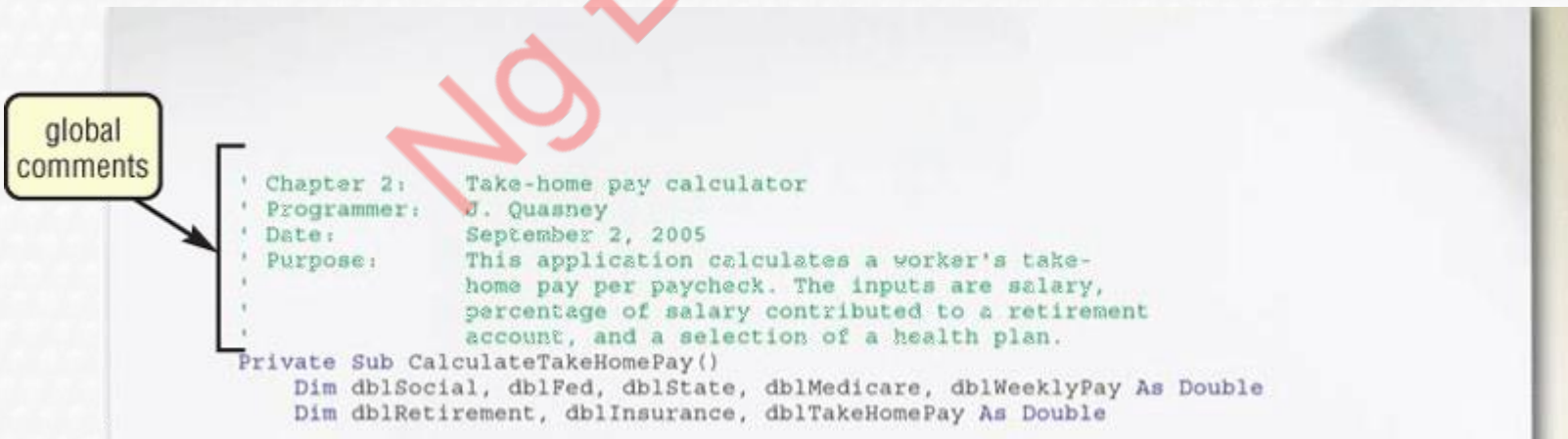
Các bước xây dựng mã nguồn (chương trình) cho 1 đề án phần mềm

Bước 4: Cài đặt (Implementation Design)

➤ **Viết code : chuyển từ thiết kế thành program**

- **Syntax**—Quy tắc xác định cách viết các lệnh
- **Comments**—Program documentation

Extreme programming (XP)—coding và testing ngay sau khi các yêu cầu được xác định



Chương 1: KỸ THUẬT TỔ CHỨC CHƯƠNG TRÌNH

■ II. Tổ chức mã nguồn cho đề án phần mềm

Các bước xây dựng mã nguồn (chương trình) cho 1 đề án phần mềm

Bước 5: Kiểm thử (Test Solution)

**Đảm bảo CT chạy thông và
cho kết quả chính xác**

**Debugging—Tìm và sửa các
lỗi syntax và logic errors**

**Kiểm tra phiên bản
beta, giao cho Users
dùng thử và thu thập
phản hồi**

Chương 1: KỸ THUẬT TỔ CHỨC CHƯƠNG TRÌNH

■ II. Tổ chức mã nguồn cho đề án phần mềm

Các bước xây dựng mã nguồn (chương trình) cho 1 đề án phần mềm

Bước 6: **Viết tài liệu** (Document Solution)

Rà soát lại program code—
loại bỏ các **dead code**, tức
các lệnh mà CT không bao
giờ gọi đến

Rà soát, hoàn thiện
Tài liệu

Chương 1: KỸ THUẬT TỔ CHỨC CHƯƠNG TRÌNH

III. Các qui tắc cơ bản (*Fundamental Rules*) trong lập trình

▪Đơn giản hóa Code – *Code Simplification* :

- Hầu hết các chương trình chạy nhanh là đơn giản. Vì vậy, hãy đơn giản hóa chương trình để nó chạy nhanh hơn. Tuy nhiên...

▪Đơn giản hóa vấn đề - *Problem Simplification*:

- Để tăng hiệu quả của chương trình, hãy đơn giản hóa vấn đề mà nó giải quyết.

▪Không ngừng nghi ngờ - *Relentless Suspicion*:

- Đặt dấu hỏi về sự cần thiết của mỗi mẫu code và mỗi thuộc tính trong cấu trúc dữ liệu.

▪Liên kết sớm - *Early Binding*:

- Hãy thực hiện xây dựng các modul dùng chung để tránh viết lặp nhiều lần sau này.

Chương 1: KỸ THUẬT TỔ CHỨC CHƯƠNG TRÌNH

IV. Tối ưu hóa mã lệnh

Các kỹ thuật tối ưu hóa:

1. Chọn giải thuật hay nhất có thể

Tìm ước số chung lớn nhất của 2 số tự nhiên a, b

Giai thuật 1:

```
int UCLN(int a, int b)
{
    while(a!=b)
    {
        if(a>b)
            a-=b;
        else
            b-=a;
    }
    return b;
}
```



Giai thuật 2:

```
int UCLN(int x, int y)
{
    int t = x % y;
    while (t!=0)
    {
        x=y;
        y=t;
        t=x % y;
    }
    return y;
}
```

Chương 1: KỸ THUẬT TỔ CHỨC CHƯƠNG TRÌNH

IV. Tối ưu hóa mã lệnh

2. Xác định những đoạn quyết định độ phức tạp thuật toán

- Vòng lặp
- Chương trình con

a. Loại bỏ biểu thức thông thường: khởi tạo các hằng chứa giá trị biểu thức

```
/*-----Bad code-----*/  
float a,b;  
for(int i=0; i<n; i++)  
{  
    a=sin(0.25);  
    b+=a*i;  
}
```

```
/*-----Good code-----*/  
float a=sin(0.25);  
float b;  
for (int i=0; i<n; i++)  
{  
    b+=a*i;  
}
```

Chương 1: KỸ THUẬT TỔ CHỨC CHƯƠNG TRÌNH

IV. Tối ưu hóa mã lệnh

2. Xác định những đoạn quyết định độ phức tạp thuật toán

- Vòng lặp
- Chương trình con

b. Tính toán trước các giá trị của các biểu thức đơn giản thường sử dụng

```
int f(int i) {  
    if (i < 10 && i >= 0)  
    {  
        return i * i - i;  
    }  
    return 0;  
}
```



```
static int[] values =  
    {0, 0, 2, 3*3-3, ..., 9*9-9};  
int f(int i) {  
    if (i < 10 && i >= 0)  
        return values[i];  
    return 0; }
```


Chương 1: KỸ THUẬT TỔ CHỨC CHƯƠNG TRÌNH

IV. Tối ưu hóa mã lệnh

2. Xác định những đoạn quyết định độ phức tạp thuật toán

c. Thực hiện các phép biến đổi số học làm giảm các phép tính

Máy tính thực hiện phép dịch bit nhanh hơn phép nhân, chia. Phép cộng trừ thực hiện mất nhiều thời gian hơn nhân và chia.

```
/*-----Bad-----*/  
if(a*8<50)  
{  
    a=a*a+4*a-5;  
}
```



```
/*-----Good-----*/  
if((a*8)<50)  
{  
    a=(a-1)*(a+5);  
}
```

Chương 1: KỸ THUẬT TỔ CHỨC CHƯƠNG TRÌNH

IV. Tối ưu hóa mã lệnh

2. Xác định những đoạn quyết định độ phức tạp thuật toán

d. Thực hiện các phép biến đổi số học làm giảm các phép tính

Máy tính thực hiện phép dịch bit nhanh hơn phép nhân, chia. Phép cộng trừ thực hiện mất nhiều thời gian hơn nhân và chia.

```
/*-----Bad-----*/  
if(a*8<50)  
{  
    a=a*a+4*a-5;  
}
```



```
/*-----Good-----*/  
if((a<<3)<50)  
{  
    a=(a-1)*(a+5);  
}
```

Chương 1: KỸ THUẬT TỔ CHỨC CHƯƠNG TRÌNH

IV. Tối ưu hóa mã lệnh

2. Xác định những đoạn quyết định độ phức tạp thuật toán

e. Giảm các quá trình lặp

```
/*-----Bad-----*/  
for (i = 0; i < n; i++) y[i] = i;
```

```
/*-----Good-----*/  
for (i = 0; i < (n % 2); i++) {y[i] = i; y[i+1] = i+1;  
}
```



Hay

```
for (; i + 1 < n; i += 2) /* no initializer */  
{  
  y[i] = i;  
  y[i+1] = i+1;  
}
```

Chương 1: KỸ THUẬT TỔ CHỨC CHƯƠNG TRÌNH

IV. Tối ưu hóa mã lệnh

2. Xác định những đoạn quyết định độ phức tạp thuật toán

f. Không dùng các vòng lặp ngắn

```
/*-----Bad code-----*/  
int a=0;  
for(i=0; i<3; i++)  
{  
    a=a*i+i  
}
```

```
/*-----Good code-----*/  
int a=0, i=0;  
a=a*i+i;  
i++;  
a=a*i+i;  
i++;  
a=a*i+i;
```


Chương 1: KỸ THUẬT TỔ CHỨC CHƯƠNG TRÌNH

IV. Tối ưu hóa mã lệnh

2. Xác định những đoạn quyết định độ phức tạp thuật toán

g. Dùng phần tử lính canh làm giảm số phép thử

```
void InsertionSort(int a[MAX], int n)
```

```
{   int i, j, x;
```

```
   for(i = 1; i < n; i++)
```

```
   {   x = a[i];
```

```
       j = i - 1;
```

```
       while (j >= 0 && a[j] > x)
```

```
       {           a[j+1] = a[j];
```

```
                   j--;
```

```
       if (j != i - 1)
```

```
           a[j+1] = x;
```

```
   }
```

```
}
```

```
for (i=2; i <= n; i++)
```

```
{   a[0] = a[i];
```

```
    x = a[i];
```

```
    j = i - 1;
```

```
    while (a[j] > x)
```

```
    { a[j+1] = a[j];
```

```
      j = j - 1; }
```

```
    a[j+1] = x;
```

Phần tử lính canh

Chương 1: KỸ THUẬT TỔ CHỨC CHƯƠNG TRÌNH

IV. Tối ưu hóa mã lệnh

2. Xác định những đoạn quyết định độ phức tạp thuật toán

g. Hạn chế dùng hàm nếu có thể: sử dụng inline cho những hàm đơn giản

Nếu 1 hàm không có loop(for, while...) thì có thể khai báo inline. Inline code sẽ được chèn vào bất cứ chỗ nào hàm được gọi, chương trình sẽ lớn hơn chút ít nhưng sẽ nhanh hơn. Tránh được việc dùng stack -4 bước khi gọi 1 hàm.

```
#include <iostream.h>
#include <math.h>
inline double delta (double a, double b)
{
    return sqrt (a * a + b * b);
}

int main ()
{
    double k = 6, m = 9;
    // 2 dòng sau thực hiện như nhau:
    cout << delta (k, m) << '\n';
    cout << sqrt (k * k + m * m) << '\n';
}
```

Cũng nên lưu ý rằng, việc inline không phải lúc nào cũng làm tăng tốc độ chạy chương trình. Lý do là mặc dù sẽ tiết kiệm được các bước gọi và thoát hàm, nhưng nếu kích thước chương trình bị tăng quá nhiều (do inline), chương trình sẽ bị chậm trong quá trình "loading" cũng nhưng sẽ xảy ra nhiều *Page faults*.

Chương 1: KỸ THUẬT TỔ CHỨC CHƯƠNG TRÌNH

IV. Tối ưu hóa mã lệnh

3. Tối ưu hóa bộ nhớ

- Chú ý tính chất biến cục bộ và biến toàn cục:

Biến cục bộ sẽ được giải phóng sau khi chương trình con kết thúc.

- Hiểu và tận dụng bộ nhớ **cache**:

*Bộ nhớ cache có dung lượng rất nhỏ, tốc độ truy xuất rất lớn(xấp xỉ bằng tốc độ của CPU) có chức năng làm giảm sự chênh lệch về mặt tốc độ truy xuất giữa bộ nhớ chính và CPU. Khi tải dữ liệu vào cache nó sẽ tải theo từng blocks (gồm **nhiều bytes dữ liệu nằm liên tiếp nhau** trong bộ nhớ chính).*

```
/*-----Bad code-----*/  
for(int j=0; j<cot; j++)  
for(int i=0; i<hang; i++)  
{  
    A[i][j]=1;  
}
```



```
/*-----Good code-----*/  
for(int i=0; i<hang; i++)  
for(int j=0; j<cot; j++)  
{  
    A[i][j]=1;  
}
```

Chương 1: KỸ THUẬT TỔ CHỨC CHƯƠNG TRÌNH

V. Thiết kế thuật toán

- Chia bài toán ra thành nhiều bài toán nhỏ hơn
- Tìm giải pháp cho từng bài toán nhỏ
- Gộp các giải pháp cho các bài toán nhỏ thành giải pháp tổng thể cho bài toán ban đầu
- Đơn giản hóa bài toán bằng cách trừu tượng hóa: làm cái gì thay vì làm như thế nào
- Ví dụ: các hàm ở mức trừu tượng
 - ▶ Hàm sắp xếp 1 mảng các số nguyên
 - ▶ Hàm nhập vào / xuất ra các ký tự: `getchar()` , `putchar()`
 - ▶ Hàm toán học : `sin(x)`, `sqrt(x)`

Chương 1: KỸ THUẬT TỔ CHỨC CHƯƠNG TRÌNH

V. Thiết kế thuật toán

■ Thiết kế Bottom-Up

(Bottom-up design)

- *Thiết kế chi tiết 1 phần*
- *Thiết kế chi tiết 1 phần khác*
- *Lặp lại cho đến hết*

■ Bottom-up design in programming

- *Viết phần đầu tiên của CT 1 cách chi tiết cho đến hết*
- *Viết phần tiếp theo của CT 1 cách chi tiết cho đến hết*
- *Lặp lại cho đến hết*

Chương 1: KỸ THUẬT TỔ CHỨC CHƯƠNG TRÌNH

V. Thiết kế thuật toán

■ Thiết kế Top-Down

(Top-down design)

- *Thiết kế toàn bộ sản phẩm một cách sơ bộ, tổng thể*
- *Tinh chỉnh cho đến khi hoàn thiện*

■ Top-down design in **programming**

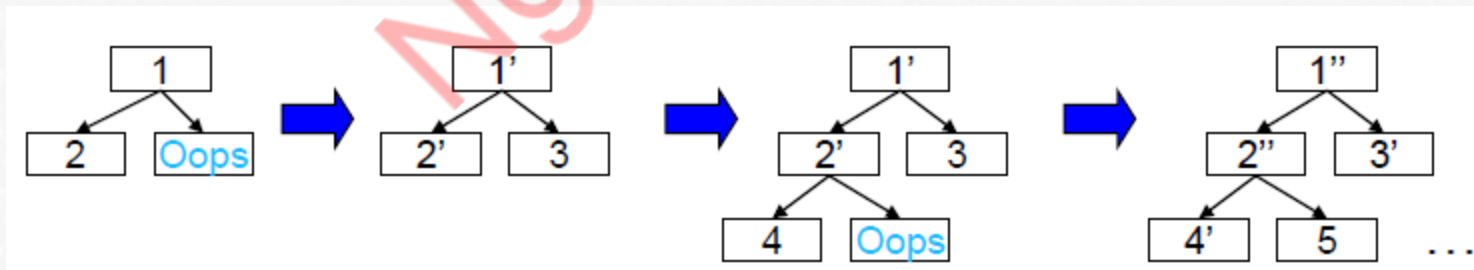
- *Phác họa hàm main() (bằng các lệnh giả ngữ - pseudocode)*
- *Tinh chỉnh từng lệnh giả ngữ*
 - Công việc đơn giản => thay bằng real code
 - Công việc phức tạp => thay bằng lời gọi hàm
- *Lặp lại sâu hơn, cụ thể, chi tiết hơn*
- *Kết quả: Sản phẩm có cấu trúc phân cấp tự nhiên*

Chương 1: KỸ THUẬT TỔ CHỨC CHƯƠNG TRÌNH

V. Thiết kế thuật toán

■ Thiết kế Top-Down

- Định nghĩa hàm main() bằng giả ngữ
- Tinh chỉnh từng lệnh giả ngữ
 - Nếu gặp sự cố: xem lại thiết kế, và...
 - Quay lại để tinh chỉnh giả ngữ đã có, và tiếp tục
- Lặp lại (trong hầu hết các trường hợp) ở mức sâu hơn, cụ thể hơn, cho đến khi các hàm được định nghĩa xong



Chương 1: KỸ THUẬT TỔ CHỨC CHƯƠNG TRÌNH

VI. Phong cách lập trình

Trong tài liệu “*Elements of Programming Style*” của Kernighan & Plauger có liệt kê các qui tắc sau:

1. Write clearly / don't be too clever – *Viết rõ ràng – đừng quá thông minh (kỳ bí)*
2. Say what you mean, simply and directly – *Tr bày vấn đề 1 cách đơn giản, trực tiếp*
3. Use library functions whenever feasible. – *Sử dụng thư viện mọi khi có thể*
4. Avoid too many temporary variables – *Tránh dùng nhiều biến trung gian*
5. Write clearly / don't sacrifice clarity for efficiency – *Viết rõ ràng / đừng hy sinh sự rõ ràng cho hiệu quả*
6. Let the machine do the dirty work – *Hãy để máy tính làm những việc nặng nhọc của nó. (tính toán ...)*
7. Replace repetitive expressions by calls to common functions. – *Hãy thay những biểu thức lặp đi lặp lại bằng cách gọi các hàm*

Chương 1: KỸ THUẬT TỔ CHỨC CHƯƠNG TRÌNH

8. Parenthesize to avoid ambiguity. – *Dùng dấu ngoặc để tránh rắc rối*
9. Choose variable names that won't be confused – *Chọn tên biến sao cho tránh được lẫn lộn*
10. Avoid unnecessary branches. – *Tránh phân nhánh không cần thiết*
11. If a logical expression is hard to understand, try transforming it – *Nếu 1 biểu thức logic khó hiểu, cố gắng chuyển đổi cho đơn giản*
12. Choose a data representation that makes the program simple – *Hãy lựa chọn cấu trúc dữ liệu để chương trình thành đơn giản*
13. Write first in easy-to-understand pseudo language; then translate into whatever language you have to use. – *Trước tiên hãy viết ct bằng giả ngữ dễ hiểu, rồi hãy chuyển sang ngôn ngữ cần thiết.*
14. Modularize. Use procedures and functions. – *Môđun hóa. Dùng các hàm và thủ tục*

Chương 1: KỸ THUẬT TỔ CHỨC CHƯƠNG TRÌNH

16. Don't patch bad code / rewrite it. – *Không chấp vá mã xấu – Viết lại đoạn code đó*
17. Write and test a big program in small pieces. – *Viết và kiểm tra 1 CT lớn thành từng CT con*
18. Use recursive procedures for recursively-defined data structures. – *Hãy dùng các thủ tục đệ qui cho các cấu trúc dữ liệu đệ qui.*
19. Test input for plausibility and validity. – *Kiểm tra đầu vào để đảm bảo tính chính xác và hợp lệ*
20. Make sure input doesn't violate the limits of the program. – *Hãy đảm bảo đầu vào không quá giới hạn cho phép của CT*
21. Terminate input by end-of-file marker, not by count. – *Hãy kết thúc dòng nhập bằng ký hiệu EOF, không dùng phép đếm*

Chương 1: KỸ THUẬT TỔ CHỨC CHƯƠNG TRÌNH

22. Identify bad input; recover if possible. – *Xác định đầu vào xấu, khôi phục nếu có thể*
23. Make input easy to prepare and output self-explanatory. – *Hãy làm cho đầu vào đơn giản, dễ chuẩn bị và đầu ra dễ hiểu*
24. Use uniform input formats. – *Hãy dùng các đầu vào theo các định dạng chuẩn.*
25. Make sure all variable are initialized before use.- *Hãy đảm bảo các biến được khởi tạo trước khi sử dụng*
26. Test programs at their boundary values. – *Hãy kiểm tra CT tại các cận*
27. Check some answers by hand. – *Kiểm tra 1 số câu trả lời bằng tay*
28. 10.0 times 0.1 is hardly ever 1.0. – *10 nhân 0.1 không chắc đã = 1.0*
29. 7/8 is zero while 7.0/8.0 is not zero. $7/8 = 0$ nhưng $7.0/8.0 \neq 0$?
30. Make it right before you make it faster. – *Hãy làm cho CT chạy đúng, trước khi làm nó chạy nhanh*

Chương 1: KỸ THUẬT TỔ CHỨC CHƯƠNG TRÌNH

31. Let your compiler do the simple optimizations. – *Hãy để trình dịch thực hiện các việc tối ưu hóa đơn giản*
32. Don't strain to re-use code; reorganize instead. – *Đừng cố tái sử dụng mã, thay vì vậy, hãy tổ chức lại mã*
33. Make sure special cases are truly special. – *Hãy đảm bảo các trường hợp đặc biệt là thực sự đặc biệt*
34. Keep it simple to make it faster. – *Hãy giữ nó đơn giản để làm cho nó nhanh hơn*
35. Make sure comments and code agree. – *Chú thích phải rõ ràng, sát code*
36. Don't comment bad code | rewrite it. – *Đừng chú thích những đoạn mã xấu, hãy viết lại*
37. Use variable names that mean something. – *Dùng các tên biến có nghĩa*

Chương 1: KỸ THUẬT TỔ CHỨC CHƯƠNG TRÌNH

38. Format a program to help the reader understand it.- *Hãy định dạng CT để giúp người đọc hiểu đc CT*

39. Don't over-comment. – *Đừng chú thích quá nhiều*

❑ Cần lưu ý rằng các quy tắc của “programming style”, giống như quy tắc văn phạm English, đôi khi bị vi phạm, thậm chí bởi những nhà văn hay nhất. Tuy nhiên khi 1 quy tắc bị vi phạm, thì thường được bù lại bằng một cái gì đó, đáng để ta mạo hiểm

❑ *Tham khảo chi tiết Slide bài giảng thầy Đặng Bình Phương*

Chương 1: KỸ THUẬT TỔ CHỨC CHƯƠNG TRÌNH

VI. GỠ LỖI

a. Gỡ lỗi (Debug)

- Gỡ rối là gì?

- *Khi chương trình bị lỗi, gỡ rối là các công việc cần làm để làm cho chương trình dịch thông, chạy thông*

- *Gỡ rối luôn là thao tác phải làm khi lập trình, thao tác này rất tốn kém*

- Cách tốt nhất vẫn là phòng ngừa

- *Chương trình không chạy? Lý do tại sao chương trình không chạy?*

- *Nếu hiểu chương trình \Rightarrow sẽ có ít sai lầm và dễ dàng sửa chữa sai sót hơn. **Bí quyết là viết mã đơn giản, hiệu quả, chú thích hợp lý.***

Chương 1: KỸ THUẬT TỔ CHỨC CHƯƠNG TRÌNH

VI. GỠ LỖI

a. Gỡ lỗi

- Lỗi là lỗi của chương trình, và trách nhiệm gỡ rối thuộc về LTV.
- Các lỗi thường gặp:
 - Lỗi từ vựng (token)
 - Lỗi cú pháp (syntax)
 - Lỗi ngữ nghĩa (symatic)
- Mọi đoạn chương trình đều có khả năng sinh lỗi
 - Lỗi chủ quan: do lập trình sai
 - Lỗi khách quan: do hệ thống

Chương 1: KỸ THUẬT TỔ CHỨC CHƯƠNG TRÌNH

VI. GỠ LỖI

b. Phòng ngừa:

- Sử dụng hàm khi lập trình
- Viết chú thích đầy đủ
- Đặt tên biến có ý nghĩa
- Soạn dữ liệu để kiểm thử
- Dùng Breakpoints
- Dùng Watch Window

Chương 1: KỸ THUẬT TỔ CHỨC CHƯƠNG TRÌNH

VI. GỠ LỖI

c. Tìm kiếm & Gỡ lỗi

Gỡ rối liên quan đến việc suy luận lùi.

Một số vấn đề không thể xảy ra và chỉ có những thông tin xác thực mới đáng tin cậy.

Phải đi ngược từ kết quả để khám phá nguyên nhân.

Khi có lời giải thích đầy đủ, ta sẽ biết được vấn đề cần sửa và có thể phát hiện ra một số vấn đề khác “



Chương 1: KỸ THUẬT TỔ CHỨC CHƯƠNG TRÌNH

VI. GỠ LỖI


d. Gỡ lỗi Heuristic

Debugging Heuristic	Áp dụng khi nào
(1) Hiểu các thông báo lỗi (error messages)	Build-time (dịch)
(2) Nghĩ trước khi viết lại chương trình	Run-time (chạy)
(3) Tìm kiếm các lỗi (bug) hay xảy ra	
(4) Divide and conquer	
(5) Viết thêm các đoạn mã kiểm tra để chương trình tự kiểm tra nó	
(6) Hiện thị kết quả	
(7) Sử dụng debugger	
(8) Tập trung vào các lệnh mới viết / mới viết lại	

Cám ơn đã theo dõi



Imperative paradigm

- Các đặc trưng chính của Mô thức này:
 - ❖ Về mặt nguyên lý và ý tưởng : *Công nghệ phần cứng và ý tưởng của J.Von Neumann*
 - ❖ Các bước tính toán, thực hiện với mục đích kiểm soát cấu trúc điều khiển. Chúng ta gọi các bước là các mệnh lệnh - *commands*
 - ❖ Những lệnh đặc trưng của imperative languages là : Assignment, IO, procedure calls
 - ❖ Các ngôn ngữ đại diện : Fortran, Algol, Pascal, Basic, C
 - ❖ Các thủ tục và hàm chính là hình ảnh về sự trừu tượng : che dấu các lệnh trong CT con, có thể coi CT con là 1 lệnh
 - ❖ Còn gọi là "Procedural programming" 

Functional paradigm

- ❖ Functional programming trên nhiều khía cạnh là đơn giản và rõ ràng hơn imperative. Vì nguồn gốc của nó là toán học thuần túy: Lý thuyết hàm. Trong khi imperative paradigm bắt nguồn từ ý tưởng công nghệ cơ bản là digital computer, phức tạp hơn, kém rõ ràng hơn lý thuyết toán học về hàm.
- ❖ Functional programming dựa trên nền tảng khái niệm toán học về hàm và 1 NNLT hàm bao gồm ít nhất những thành phần sau :
 - Tập hợp các cấu trúc dữ liệu và các hàm liên quan
 - Tập hợp các hàm cơ sở - Primitive Functions.
 - Tập hợp các toán tử .

Functional paradigm

- Các đặc trưng cơ bản :

- ❖ Về mặt nguyên lý và ý tưởng : Toán học và lý thuyết hàm
- ❖ Các giá trị tạo được là không thể biến đổi *non-mutable*
- ❖ Không thể thay đổi các yếu tố của giá trị hợp thành
- ❖ Giống như phương thuốc, có thể tạo một phiên bản của các giá trị hợp thành : một giá trị trung gian
- ❖ Trừu tượng 1 biểu thức đơn thành 1 hàm và hàm có thể tính toán như là 1 biểu thức
- ❖ Các hàm là những giá trị đầu tiên
- ❖ Hàm là dữ liệu hoàn chỉnh, giống như số, danh sách, ...
- ❖ Thích hợp với xu hướng tính toán theo yêu cầu
- ❖ Mở ra những khả năng mới

Ví dụ về lập trình hàm

```
function GT(n: longint) : longint;  
  var x : longint;  
  Begin  
    x:=1;  
    while (n > 0) do begin  
      x := x * n;  
      n := n- 1;  
    end;  
    GT := x;  
  End;
```


```
Function GT(n: longint) : Longint;  
Begin  
  if n=1 then GT :=1  
  else GT := n* GT(n-1);  
End;
```

Với functional paradigm, ta có thể viết

```
GT n =  
  if n = 1 then 1  
  else n * GT(n -1);
```



Logic paradigm


- ❖ Mô thức lập trình logic hoàn toàn khác với các Mô thức còn lại.
- ❖ Mô thức này đặc biệt phù hợp với những lĩnh vực liên quan đến việc rút ra những kiến thức từ những sự kiện và quan hệ cơ bản – lĩnh vực trí tuệ nhân tạo. Mô thức này không gắn với những lĩnh vực tính toán nói chung.
- ❖ Trả lời 1 câu hỏi thông qua việc tìm các giải pháp
- ❖ Các đặc trưng:
 - Về nguyên tắc và ý tưởng: Dựa trên các tiên đề - axioms, các quy luật suy diễn - inference rules, và các truy vấn - queries.
 - Chương trình thực hiện từ việc tìm kiếm có hệ thống trong 1 tập các sự kiện, sử dụng 1 tập các luật để đưa ra kết luận. 

object-oriented paradigm

- Mô thức hướng đối tượng thu hút được sự quan tâm và nổi tiếng từ khoảng 20 năm nay. Lý do là khả năng hỗ trợ mạnh của tính bao gói và gộp nhóm logic của các khía cạnh lập trình. Những thuộc tính này rất quan trọng khi mà kích cỡ các chương trình ngày càng lớn.
- Nguyên nhân cơ bản và sâu sắc dẫn đến thành công của Mô thức này là :
 - Cơ sở lý thuyết đỉnh cao của Mô thức. Một CT HĐT được xây dựng với những khái niệm, tư tưởng làm cơ sở, điều đó rất quan trọng và theo cách đó tất cả các kỹ thuật cần thiết cho lập trình trở thành thứ yếu.
- Gửi thông điệp giữa các objects để mô phỏng sự tiến triển theo thời gian của hàng loạt các hiện tượng trong thế giới thực.

object-oriented paradigm

▪ Các đặc trưng

- Nguyên lý và ý tưởng : Lý thuyết về khái niệm - concepts, và các Mô thức tương tác trong thế giới thực
- Dữ liệu cũng như các thao tác trên dữ liệu được đóng gói trong objects
- Cơ chế che dấu thông tin được sử dụng để tránh những tác động từ bên ngoài object
- Các Objects tương tác với nhau qua việc truyền thông điệp, đó là phép ẩn dụ cho việc thực hiện các thao tác trên 1 object
- Trong phần lớn các NNLT HĐT, objects được nhóm lại trong classes
 - **Objects trong classes có chung các thuộc tính, cho phép lập trình trên lớp, thay vì lập trình trên từng đối tượng riêng lẻ**
 - **Classes đại diện cho concepts còn objects đại diện cho hiện tượng**
 - **Classes được tổ chức trong cây phả hệ có kế thừa**
 - **Tính kế thừa cho phép mở rộng hay chuyên biệt hóa lớp **