

Programming Technique



KTLT (3t)

Programming technique

SE Dept
<SE-SoICT-HUT>
Lương Mạnh Bá
balm@soict.hut.edu.vn

Programming Technique

Programming Languages – classifications and Program Development



Nội dung

NNLT và
phân loại NNLT

Develop Web pages : HTML, scripting languages,
DHTML, XML, WML, và các phần mềm tạo
trang web

c

Multimedia
authoring programs

c quan (Visual programming languages)

6 bước của chu trình phát triển phần mềm
Six steps in the program development cycle

ng (Object-oriented
programming languages)

(Nonprocedural
languages and tools)

ng

Những cấu trúc cơ sở dùng thiết kế chương trình

Last update 8-2010

CuuDuongThanCong.com

SE-SoICT

<https://fb.com/tailieudientucnlt>

KT LT-1.3

Next

Chương trình máy tính và ngôn ngữ lập trình

Computer Programs and Programming Languages

Computer program?

- **Tập hợp các lệnh chỉ dẫn cho máy tính thực hiện nhiệm vụ**
 - **Programming language**—Dùng để viết các lệnh, chỉ thị



Last update 8-2010

CuuDuongThanCong.com

SE-SoICT

<https://fb.com/tailieudientucong>

KTLT 1.1
Next ➤

programming language - NNLT

Một NNLT 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. Nhiệm vụ được thực thi gọi là một computation, tuân thủ một độ chính xác và những quy tắc nhất quán.

Với **mỗi ngôn ngữ lập trình**, ta cần nắm bắt, thấu hiểu những gì ?: Có 3 thành phần căn bản của bất cứ 1 NNLT nào.

Mô thức ngôn ngữ-Language paradigm là những nguyên tắc chung cơ bản, dùng bởi LTV để xây dựng chương trình.

Cú pháp - Syntax là cách để xác định những gì là hợp lệ trong cấu trúc các

LT. Tuy nhiên điều đó không có nghĩa là nó giúp chúng ta hiểu hết ý nghĩa của câu văn.

Ngữ nghĩa – semantics

a là 1 thành phần không thể thiếu của 1 ngôn ngữ.

Có rất nhiều NNLT, 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.

Cont...

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)

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

- Tính mềm dẻo 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

HOẠT ĐỘNG CỦA 1 CHƯƠNG TRÌNH

- ❖ Computer program được nạp vào nh
như là 1 tập các lệnh bằng ngôn ngữ máy, tức
là một dãy tuần tự các số nhị phân - *binary digits*.
- ❖ Tại bất cứ một thời điểm nào, computer sẽ ở
một trạng thái -state nào đó.
- ❖ Đặc điểm cơ bản của trạng thái là con trỏ lệnh *instruction pointer* trỏ tới lệnh tiếp theo để thực hiện.
- ❖ Thứ tự thực hiện các nhóm lệnh mã máy
được gọi là luồng điều khiển *flow of control*.

MACHINE CODE

- ❖ Máy tính chỉ nhận các tín hiệu điện tử - có, không có - tương ứng với các dòng bits.
- ❖ 1 CT ở dạng đó gọi là *machine code*.
- ❖ Ban đầu chúng ta phải dùng machine code để viết CT:
- ❖ Quá phức tạp, giải quyết các bài toán lớn là không tưởng

23fc 0000 0001 0000 0040
0cb9 0000 000a 0000 0040
6e0c
06b9 0000 0001 0000 0040
60e8

ASSEMBLY LANGUAGE

- ❖ NN Assembly là bước đầu tiên của việc xây dựng cơ chế viết chương trình tiện lợi hơn – thông qua các ký hiệu, từ khóa và cả mã máy.
- ❖ Tất nhiên, để chạy được các chương trình này thì phải dịch (assembled) thành machine code.
- ❖ Vẫn còn phức tạp, cải thiện không đáng kể

```
        movl    #0x1 , n  
compare:  
        cmpl    #0xa , n  
        cgt     end_of_loop  
        acddl   #0x1 , n  
        bra     compare  
end_of_loop:
```

HIGH LEVEL LANGUAGE

- ❖ Thay vì dựa trên phần cứng (machine-oriented) cần tìm cơ chế dựa trên vấn đề (problem-oriented) để tạo chương trình.
- ❖ Chính vì thế *high(er) level languages* – là các ngôn ngữ lập trình gần với ngôn ngữ tự nhiên hơn – dùng các từ khóa giống tiếng anh – đã được xây dựng như : Algol, Fortran, Pascal, Basic, Ada, C, ...

cuu duong than cong . com

PHÂN LOẠI THEO THỜI GIAN

- ❖ 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 Bắt đầu cuộc khủng hoảng phần mềm “software crisis”:
 1. Giảm sự phụ thuộc vào máy – Tính chuyển đổi.
 2. Tăng sự đúng đắn của CT-Structured Programming, modular programming và information hiding.

Ví dụ : Pascal, Algol 68 and C.

Continue ...

- ❖ 1980s Giảm sự phức tạp – object orientation, functional programming.
- ❖ 1990s Khai thác phần cứng song song và phân tán (parallel và distributed) làm cho chương trình chạy nhanh hơn, kết quả là hàng loạt ngôn ngữ mở rộng khả năng lập trình parallel cũng như các NNLT chuyên parallel như occam được xây dựng.
- ❖ 2000s Genetic programming languages, DNA computing, bio-computing?
- ❖ Trong tương lai : Ngôn ngữ LT lượng tử : Quantum ?

SOFTWARE CRISIS

Khái niệm software crisis bao gồm hàng loạt vấn đề nảy sinh trong việc phát triển phần mềm trong những năm 1960s khi muốn xây dựng những hệ thống phần mềm lớn trên cơ sở các kỹ thuật phát triển thời đó.

Kết quả:

- 1. Thời gian và giá thành tăng vọt tới mức không thể chấp nhận nổi.
- 2. Năng suất của các LTV không đáp ứng yêu cầu.
- 3. Chi phí LTV
- 4. Chất lượng phần mềm bị giảm, thấp.

Để giải quyết các vấn đề kể trên, chuyên ngành software engineering (SE) ra đời.

CÁC THẾ HỆ NNLT LANGUAGE GENERATIONS

Generation	Classification
1st	Machine languages
2nd	Assembly languages
3rd	Procedural languages
4th	Application languages (4GLs)
5th	AI techniques, inference languages
6th	Neural networks (?), others....

Computer Programs and Programming Languages

Low-level languages và high-level languages?

Low-level language

High-level language

Machine-dependent

Phụ thuộc phần cứng, chỉ chạy trên
một loại máy tính

Machine-independent

Thường không phụ thuộc phần
cứng, có thể chạy trên nhiều loại
máy tính khác nhau

Machine và assembly languages là
ngôn ngữ bậc thấp (low-level)

PHÂN LOẠI THEO MỨC ĐỘ TRÙU TƯỢNG

Level	Instructions	Memory handling
Low level languages	Dạng bits – giống các lệnh	Truy cập và cấp phát trực tiếp bộ nhớ
High level languages	Dùng các biểu thức và các dòng điều khiển	Truy cập và cấp phát bộ nhớ qua các lệnh, toán tử -
Very high level languages	Hoàn toàn trùu tượng, độc lập phần cứng	Che dấu hoàn toàn việc truy cập và tự động cấp phát bộ nhớ

DECLARATIVE và NON-DECLARATIVE PROGRAMMING

- ❖ Các ngôn ngữ có thể chia thành 2 nhóm :
- ❖ Nhóm 1 gọi là Declarative (tường thuật - chính là functional và logic languages).
- ❖ Nhóm 2 gọi là Non-declarative hay procedural (tức là các ngôn ngữ thủ tục, mệnh lệnh).

cuu duong than cong . com

Procedural Languages – Ngôn ngữ thủ tục

Procedural language?

i gì cần làm và làm
như thế nào?

Còn gọi là **third-generation language**
(3GL)

Sử dụng hàng loạt các từ
giống tiếng Anh để viết
các chỉ thị - instructions

Các ngôn ngữ thông
dụng: BASIC, COBOL,
PASCAL, C,C++ và JAVA



Click to view animation

Procedural Languages

Trình dịch - Compiler?

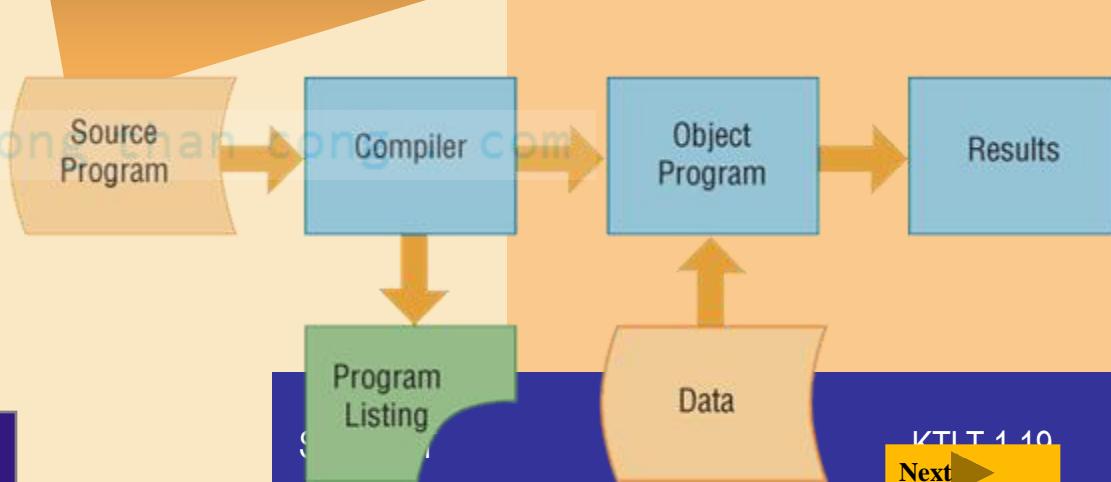
- Là chương trình thực hiện biên dịch toàn bộ các lệnh của chương trình nguồn thành mã máy trước khi thực hiện

```
* COMPUTE REGULAR TIME PAY
  MULTIPLY REGULAR-TIME-HOURS BY HOURLY-PAY-RATE
  GIVING REGULAR-TIME-PAY.

* COMPUTE OVERTIME PAY
  IF OVERTIME-HOURS > 0
    COMPUTE OVERTIME-PAY = OVERTIME-HOURS * 1.5 * HOURLY-PAY-RATE
  ELSE
    MOVE 0 TO OVERTIME-PAY.

* COMPUTE GROSS PAY
  ADD REGULAR-TIME-PAY TO OVERTIME-PAY
  GIVING GROSS-PAY.

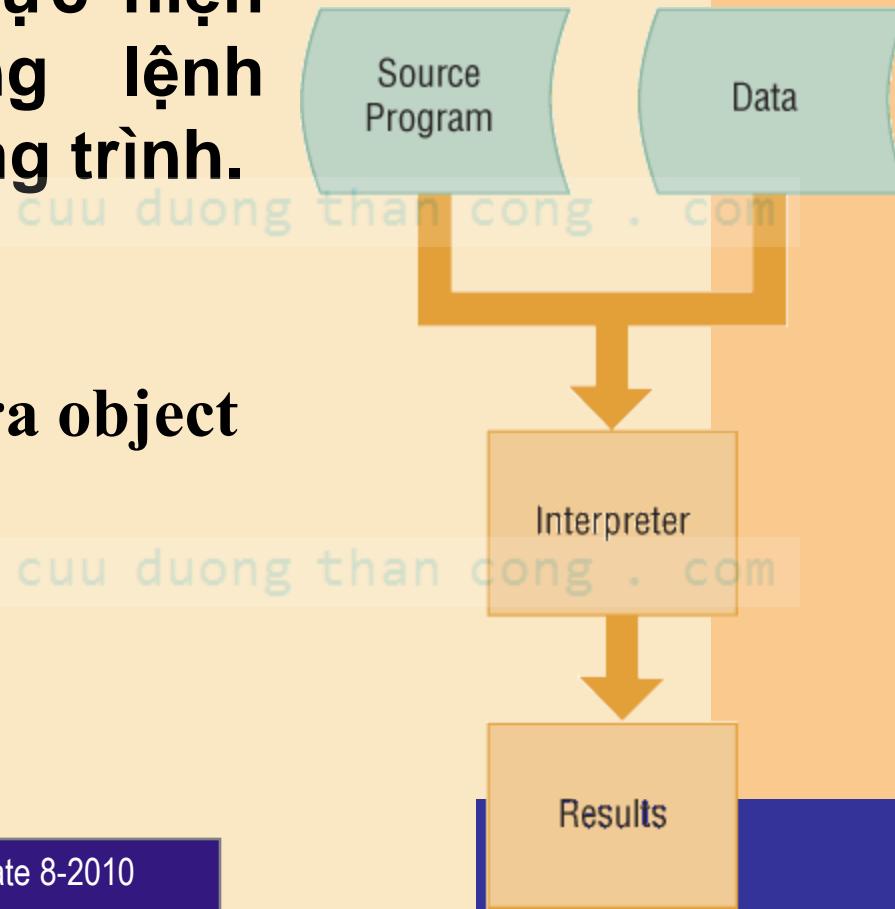
* PRINT GROSS PAY
  MOVE GROSS-PAY TO GROSS-PAY-OUT.
  WRITE REPORT-LINE-OUT FROM DETAIL-LINE
  AFTER ADVANCING 2 LINES.
```



Procedural Languages

Thông dịch - Interpreter?

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



Procedural Languages

BASIC?

- Được thiết kế để cho những người mới học tiếp cận một cách đơn giản NNLT
- Beginner's All-purpose Symbolic Instruction Code

```
REM COMPUTE REGULAR TIME PAY  
Regular.Time.Pay = Regular.Time.Hours * Hourly.Pay.Rate  
  
REM COMPUTE OVERTIME PAY  
If Overtime.Hours > 0 THEN  
    Overtime.Pay = Overtime.Hours * 1.5 * Hourly.Pay.Rate  
ELSE  
    Overtime.Pay = 0  
END IF  
  
REM COMPUTE GROSS PAY  
Gross.Pay = Regular.Time.Pay + Overtime.Pay  
REM PRINT GROSS PAY  
PRINT USING "The gross pay is $##,###.##"; Gross.Pay
```

Procedural Languages

COBOL?

- Dùng cho các ứng dụng trong kinh tế
- Các lệnh giống tiếng anh làm cho code dễ đọc, viết và chỉnh sửa
- COmmon Business-Oriented Language

```
* COMPUTE REGULAR TIME PAY  
MULTIPLY REGULAR-TIME-HOURS BY HOURLY-PAY-RATE  
GIVING REGULAR-TIME-PAY.  
  
* COMPUTE OVERTIME PAY  
IF OVERTIME-HOURS > 0  
    COMPUTE OVERTIME-PAY = OVERTIME-HOURS * 1.5 * HOURLY-PAY-RATE  
ELSE  
    MOVE 0 TO OVERTIME-PAY.  
  
* COMPUTE GROSS PAY  
ADD REGULAR-TIME-PAY TO OVERTIME-PAY  
GIVING GROSS-PAY.  
  
* PRINT GROSS PAY  
MOVE GROSS-PAY TO GROSS-PAY-OUT.  
WRITE REPORT-LINE-OUT FROM DETAIL-LINE  
AFTER ADVANCING 2 LINES.
```

Procedural Languages

C?

- Là NNLT rất mạnh, ban đầu được thiết kế để lập trình hệ thống - write system software
- Yêu cầu những kỹ năng lập trình chuyên nghiệp

```
/* Compute Regular Time Pay */  
rt_pay = rt_hrs * pay_rate;  
  
/* Compute Overtime Pay */  
if (ot_hrs > 0)  
    ot_pay = ot_hrs * 1.5 * pay_rate;  
else  
    ot_pay = 0;  
  
/* Compute Gross Pay */  
gross = rt_pay + ot_pay;  
  
/* Print Gross Pay */  
printf("The gross pay is %d\n", gross);
```

Object-Oriented Programming Languages

Object-oriented programming (OOP) language?

Dùng để hỗ trợ
thiết kế HĐT
object-oriented
design

Lợi ích cơ bản
là khả năng tái
sử dụng -
reuse existing
objects

Event-driven—
Hướng sự kiện
Kiểm tra để trả
lời một tập các
sự kiện

C++ và Java
là các NN hoàn
toàn HĐT
object-oriented
languages

Object là
phân tử chứa
đựng cả dữ
liệu và các
thủ tục xử lý
dữ liệu

Event là hành
động mà
chương trình
cần đáp ứng

Object-Oriented Programming Languages

C++?

- Chứa đựng các thành phần của C, loại bỏ những nhược điểm và thêm vào những tính năng mới để làm việc với object-oriented concepts
- Được dùng để phát triển các Database và các ứng dụng Web

```
// portion of a C++ program that allows users to create a new zip code from a
// string or a number and expand zip codes, as appropriate, to a 10-digit number
ZipC::ZipC( const unsigned long zipnum )
{
    ostringstream strInt;
    strInt << zipnum;
    code = strInt.str();
}

const string ZipC::getCode()
{
    return code;
}

void ZipC::setCode(const string newCode)
{
    code = newCode;
}

void ZipC::expand( const string suffix )
{
    if(code.length() == 5 &&           // small size?
        suffix.length() == 4)          // length ok?
    {
        code += "-";
        code.append(suffix);
    }
}
```

Last update 8-2010

CuuDuongThanCong.com

OLE-FOOTER

<https://fb.com/tailieudientucnit>

KTLEIT-TZU

Next ➤

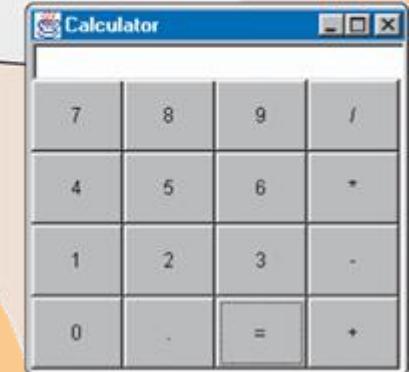
Object-Oriented Programming Languages

Java?

- Phát triển bởi Sun Microsystems
- Giống C++ nhưng dùng trình dịch just-in-time (JIT) để chuyển source code thành machine code

```
public void actionPerformed(ActionEvent e)
{
    foundKey = false;

    //Search for the key pressed
    for (int i = 0; i < keysArray.length && !foundKey; i++)
        if(e.getSource() == keysArray[i])    //key match found
    {
        foundKey = true;
        switch(i)
        {
            case 0: case 1: case 2: case 3: case 4: //number buttons
            case 5: case 6: case 7: case 8: case 9: //0 - 9
            case 15:                                         //decimal point button
                if(clearText)
                {
                    lcdField.setText("");
                    clearText = false;
                }
                lcdField.setText(lcdField.getText() + keysArray[i].getLabel());
                break;
        }
    }
}
```



Object-Oriented Programming Languages

Visual programming language?

Visual programming environment (VPE)
Cho phép developers kéo và thả các objects để xd programs

Cung cấp giao diện trực quan hoặc đồ họa để tạo source code

Đôi khi được gọi là fifth-generation language

Thường được dùng trong môi trường RAD (rapid application development)

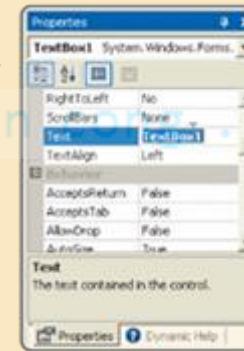
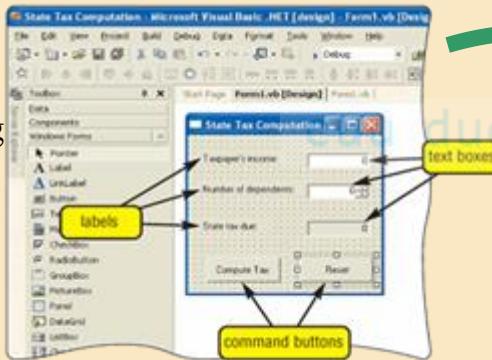
LTV viết và phát triển chương trình trong các segments

Object-Oriented Programming Languages

Visual Studio .NET 2003, 2005?

- Bước phát triển của visual programming languages và RAD tools
- .NET là tập hợp các công nghệ cho phép program chạy trên Internet
- Visual Basic .NET 2003-5 dùng để xd các ct hướng đối tượng phức tạp

Step 1. LTV
thiết kế giao
diện người dùng
-
user interface.



Step 2. LTV gán các
thuộc tính cho mỗi
object trên form.



Step 4. LTV kích hoạt

CuuDuongThanCong.com

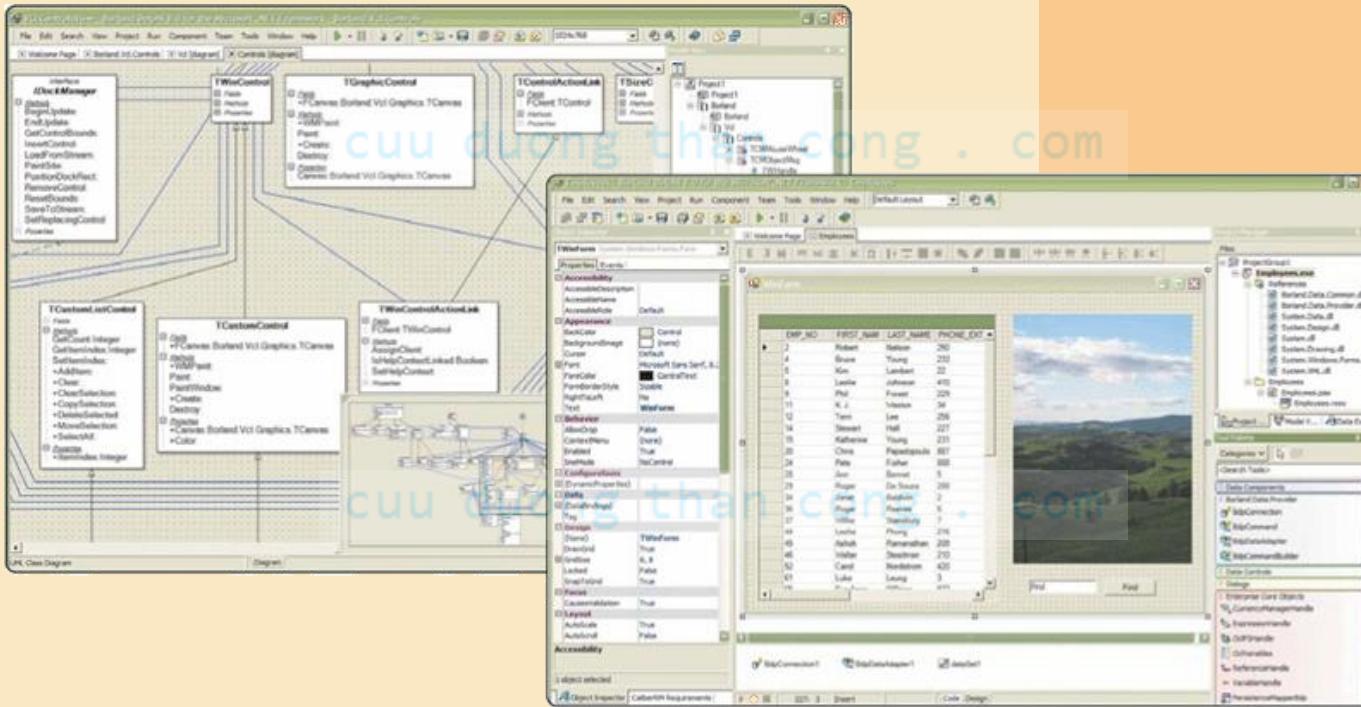
Step 3. LTV
viết code để xác
định các action
cần thực hiện đối
với các sự kiện
cần thiết.

```
1 Public Class Form1
2     Inherits System.Windows.Forms.Form
3
4     Windows Form Designer generated code
5     ' Chapter 3: State Tax Computation
6     ' Programmer: J. Quasney
7     ' Date: September 2, 2005
8     ' Purpose: This project calculates state income tax due
9     ' on income level and number of dependents.
10
11 Private Sub btnCompute_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
12     ' Calculate tax and display the result in the txtTaxDue
13     txtTaxDue.Text = 0.03 * (txtIncome.Text - 600 * nudDepen
14 End Sub
15
16 Private Sub btnReset_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
17     ' Set all input and output display values to 0
18     txtIncome.Text = "0"
19     nudDependents.Value = 0
20     txtTaxDue.Text = "0"
21 End Sub
22 End Class
23
```

Object-Oriented Programming Languages

Delphi?

- Là 1 công cụ lập trình trực quát mạnh
- Hợp với những ứng dụng chuyên nghiệp và Web lớn



Last update 8-2010

CuuDuongThanCong.com

SE-SoICT

<https://fb.com/tailieudientuclnt>

KTLT-1.29

Next ➤

Object-Oriented Programming Languages

PowerBuilder?

- Một công cụ lập trình trực quan mạnh khác
- Phù hợp với các ứng dụng Web-based hay các ứng dụng lớn HĐT - object-oriented applications



Last update 8-2010

CuuDuongThanCong.com

SE-SoICT

<https://fb.com/tailieudientucong>

KTLT-1.30

Next ➤

Nonprocedural Languages and Program Development Tools

- **nonprocedural languages và program development tools?**

Nonprocedural Language

LTV viết các lệnh giống tiếng anh hoặc tương tác với môi trường trực quan để nhận được các dữ liệu từ files hay database

Program Development Tools

Các chương trình thân thiện với người sử dụng được thiết kế để trợ giúp cả LTV lẫn người sử dụng trong việc tạo chương trình

Nonprocedural Languages and Program Development Tools

RPG (Report Program Generator)?

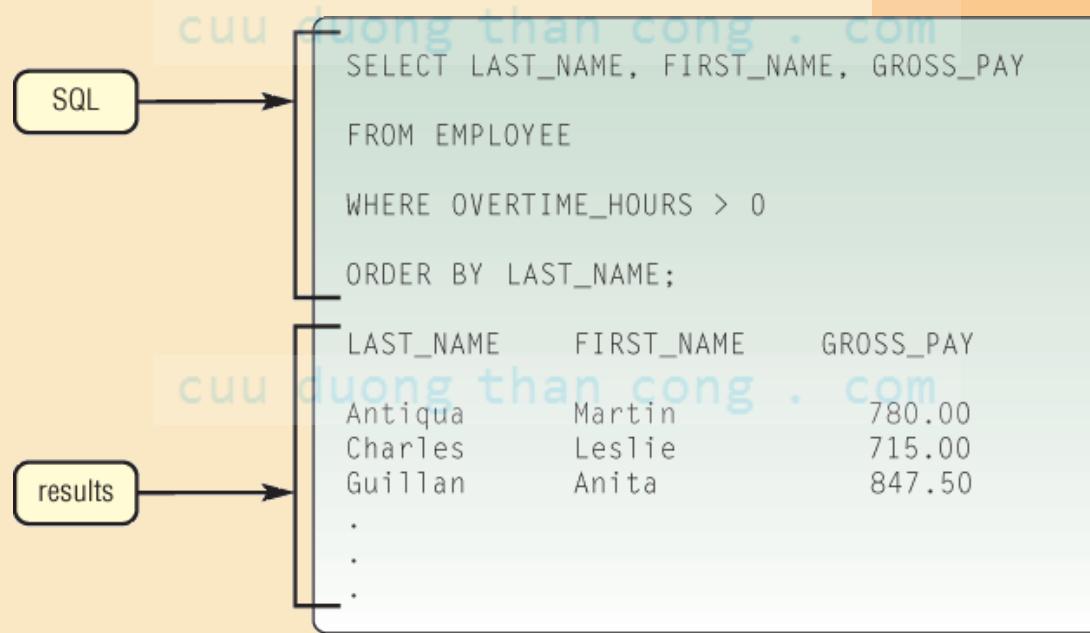
- Các ngôn ngữ LT phi thủ tục dùng để tạo các báo cáo, thiết lập các thao tác tính toán và cập nhật files

```
C* COMPUTE REGULAR TIME PAY
C      RTHRS      MULT RATE          RTPAY      72
C*
C* COMPUTE OVERTIME PAY
C      OTHRS      IFGT 0           OTRATE      72
C      RATE       MULT 1.5          OTPAY      72
C      OTRATE     MULT OTHRS
C                  ELSE
C                  INZ            OTPAY      72
C
C* COMPUTE GROSS PAY
C      RTPAY      ADD  OTPAY          GRPAY      72
C
C* PRINT GROSS PAY
C      EXCPTDETAIL
C
C*
O* OUTPUT SPECIFICATIONS
OOPRINT E      DETAIL
O          23 'THE GROSS PAY IS $'
O          GRPAY J      34
```

Nonprocedural Languages and Program Development Tools

NN thế hệ IV - fourth generation language (**4GL**)?

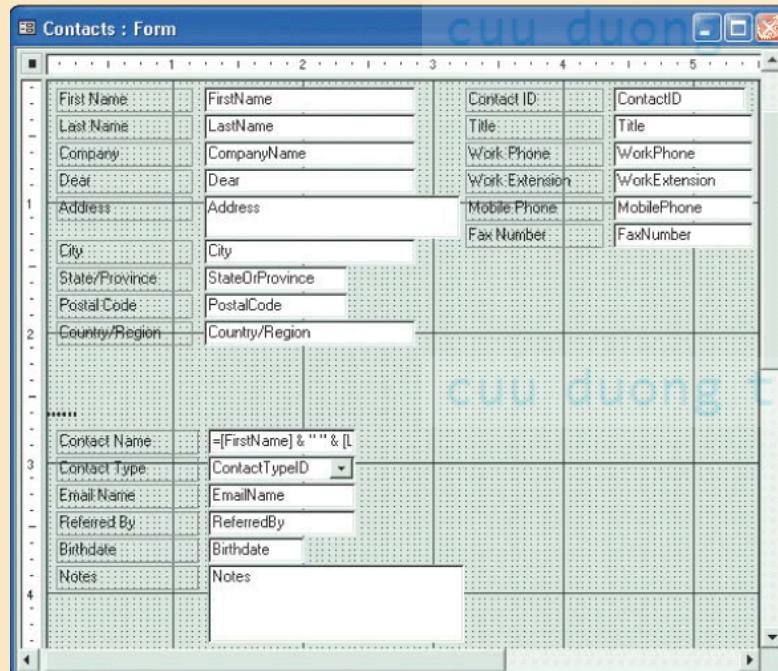
- Là các ngôn ngữ phi thủ tục cho phép truy cập dữ liệu trong CSDL.
- NNLT 4GL thông dụng là **SQL, Access**, là các ngôn ngữ truy vấn... Cho phép ND quản trị dữ liệu trong CSDL quan hệ.



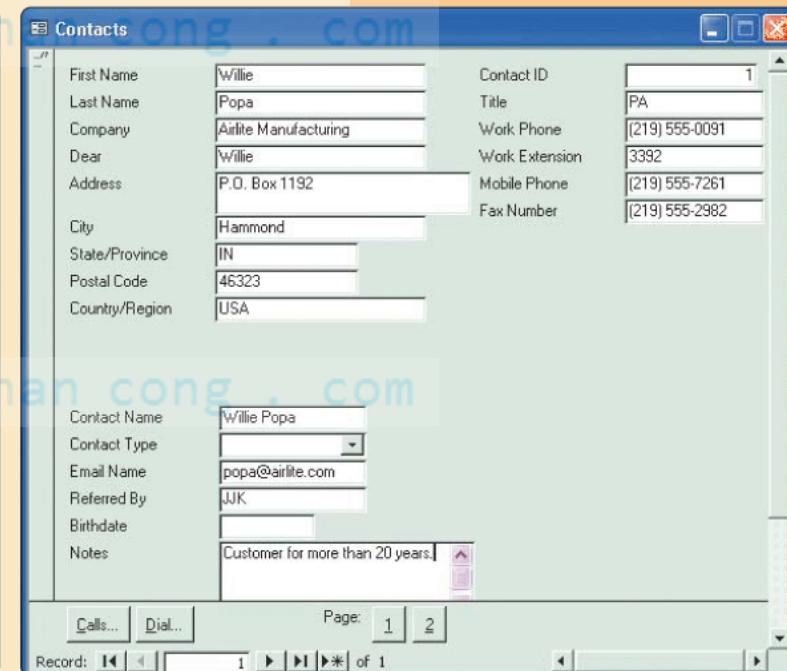
Nonprocedural Languages and Program Development Tools

Application generator?

- Là chương trình tạo mã nguồn hoặc mã máy từ các đặc tả.
- Bao gồm các chương trình tạo Report , form, và tạo menu
 - Form cung cấp các vùng để vào dữ liệu



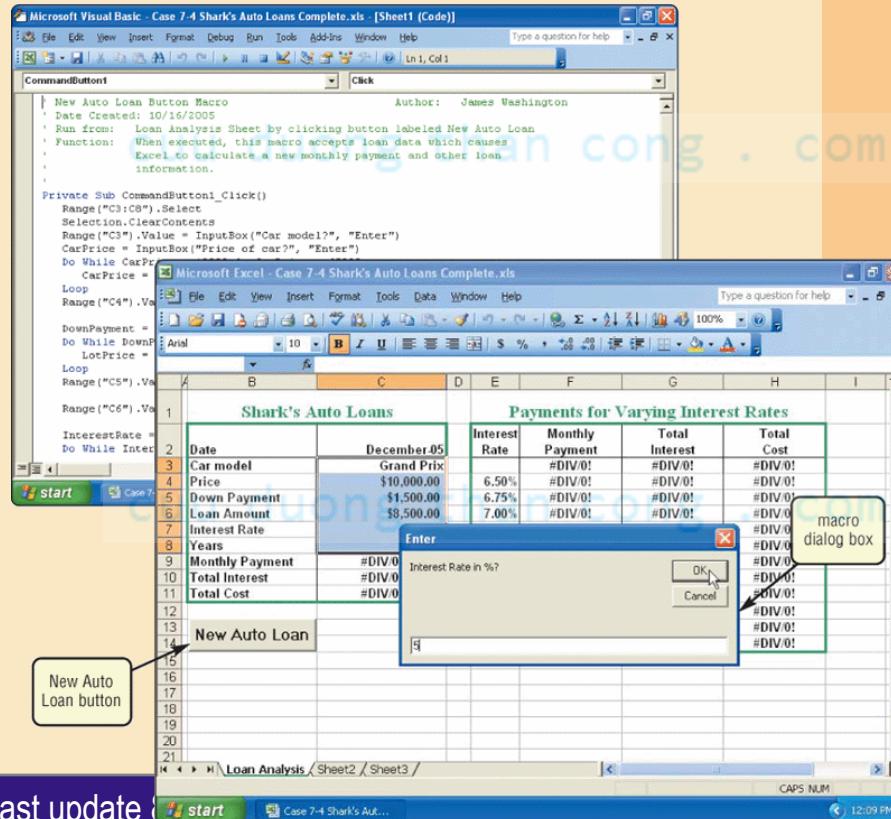
Last update 8-2010



Nonprocedural Languages and Program Development Tools

Visual Basic for Applications (VBA)?

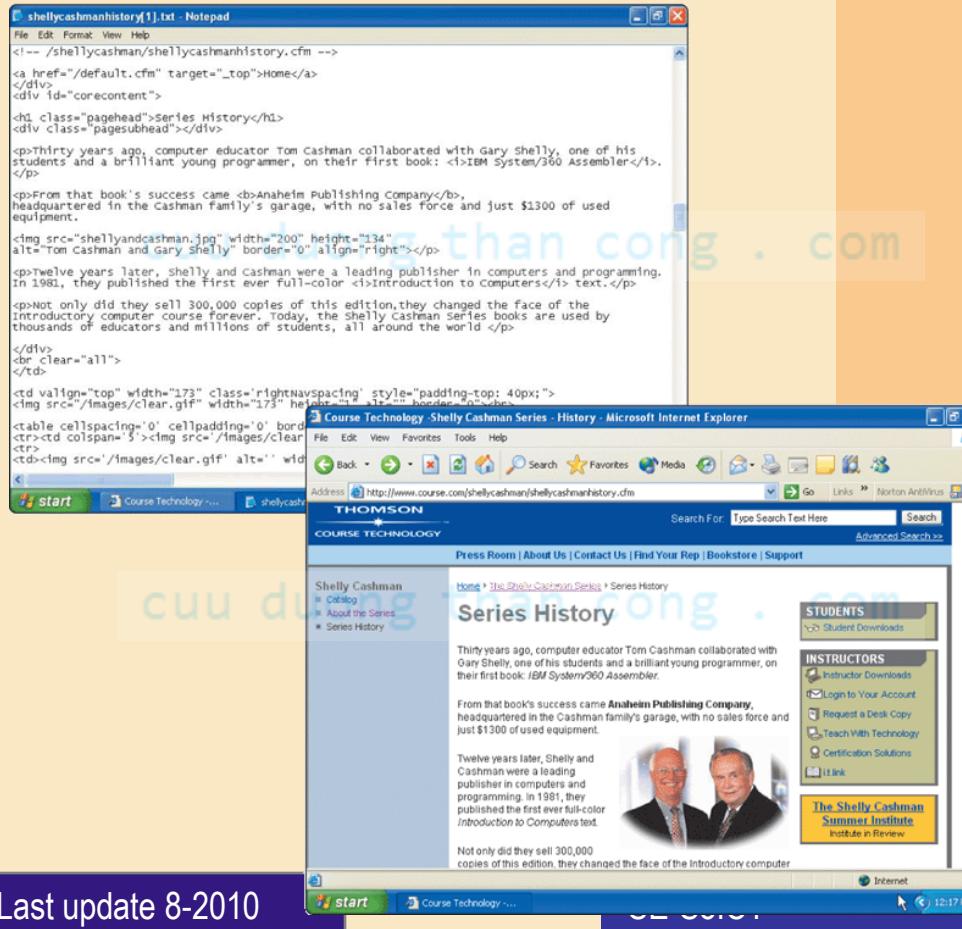
- Macro programming language
 - **Macro**—Dãy các lệnh dùng để tự động hóa các công việc



Web Page Development

HTML (Hypertext Markup Language)?

- Dùng để tạo các trang Web



Last update 8-2010

Web Page Development

Các hiệu ứng đặc biệt và các phần tử tương tác được thêm vào trang Web như thế nào ?

Script

Thông dịch chương trình chạy trên client

Applet

thường chạy trên client, nhưng được biên dịch

Servlet

applet chạy trên server

ActiveX control

Là chương trình nhỏ chạy trên client

Counter

đếm số người thăm Web site

Image map

Hình ảnh đồ họa trả tới URL

Processing form

Thu thập số liệu từ visitors

Web Page Development

Common gateway interface (CGI)?

- Chuẩn giao tiếp xác định cách thức Web server giao tiếp với các nguồn tài nguyên bên ngoài
 - CGI script—program quản trị việc gửi và nhận dữ liệu qua CGI

Step 1. LTV lưu các CGI program trong 1 thư mục đặc biệt trên Web server ví dụ /cgi-bin.



Step 2. Webmaster tạo 1 liên kết giữa CGI program và Web page. Khi 1 user hiện trang Web , CGI program sẽ automatically starts.



Step 4. CGI program nhận thông tin từ database, kết hợp chúng dưới dạng HTML , và gửi cho trình duyệt Web của User.



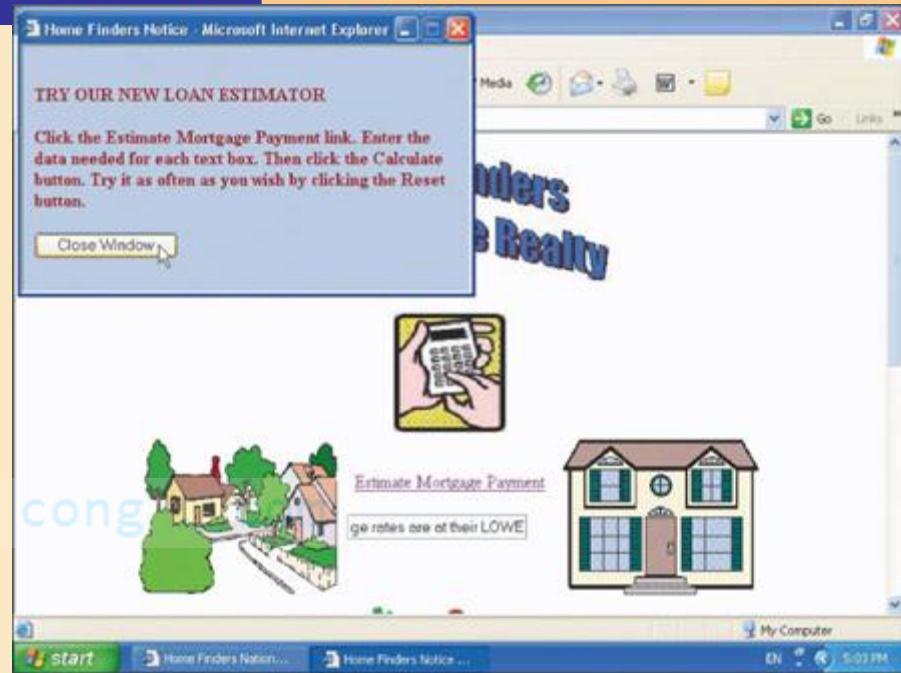
SE-SolCT

Web Page Development

Scripting language?

➤ Rất dễ học và dễ sử dụng

- **JavaScript**—thêm các nội dung động và các phần tử tương tác vào Web page
- **VBScript** (Visual Basic, Scripting Edition)—Thêm tính thông minh và tương tác vào Web page
- **Perl** (Practical Extraction and Report Language)—Có khả năng xử lý văn bản rất mạnh

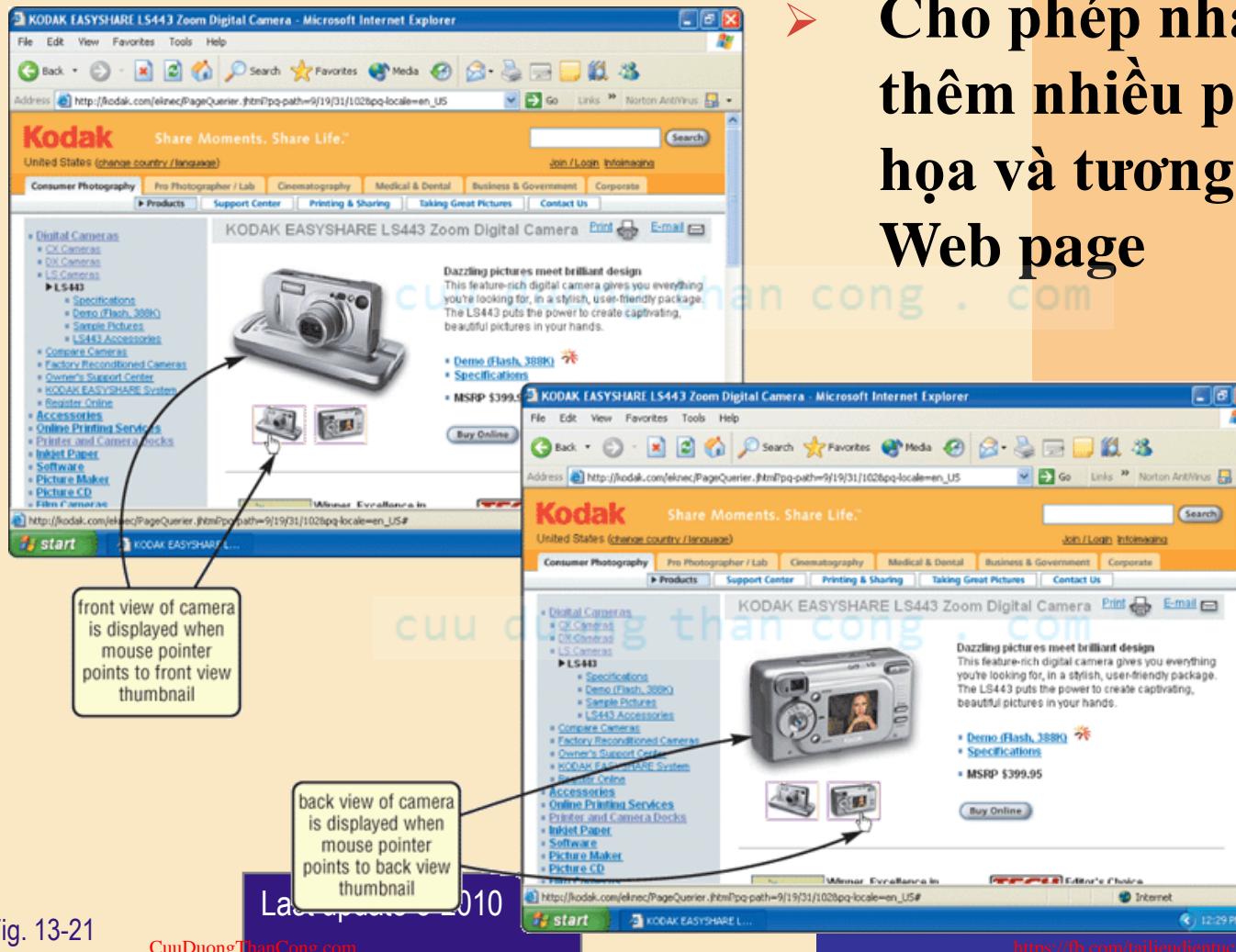


```
notice - Notepad
File Edit Format View Help
<HTML>
<HEAD>
<TITLE>Home Finders Notice</TITLE>
</HEAD>
<BODY bgcolor="#B0C4DE">
<P><BR>
<FONT COLOR="8b0000"><B>TRY OUR NEW LOAN ESTIMATOR</B></FONT>
<P>Click the Estimate Mortgage Payment link. Enter the data needed for each text box. Then click the calculate button. Try it as often as you wish by clicking the Reset button.</P>
<FORM>
<INPUT Type="Button" Value="Close window" onclick="window.close()"> </P>
</FORM>
</FONT>
</BODY>
</HTML>
```

Last update 8-2010

Web Page Development

Dynamic HTML (DHTML)?



p. 682 Fig. 13-21

CuuDuongThanCong.com

Cho phép nhà phát triển thêm nhiều phần tử đồ họa và tương tác vào Web page

KLT-1.40

Next

Web Page Development

XHTML, XML, và WML?

XHTML

(Extensible HTML)

tạo khả năng Web sites có thể hiện dễ ràng hơn trên các trình duyệt

Chứa các tính năng của HTML và XML

XML

(Extensible Markup Language)

Cho phép developers có thể tạo các thẻ - tags – riêng của mình

Server gửi toàn bộ bản ghi cho client, tạo khả năng cho client có thể thực hiện việc xử lý mà không phải quay lại server

WML

(Wireless Markup Language)

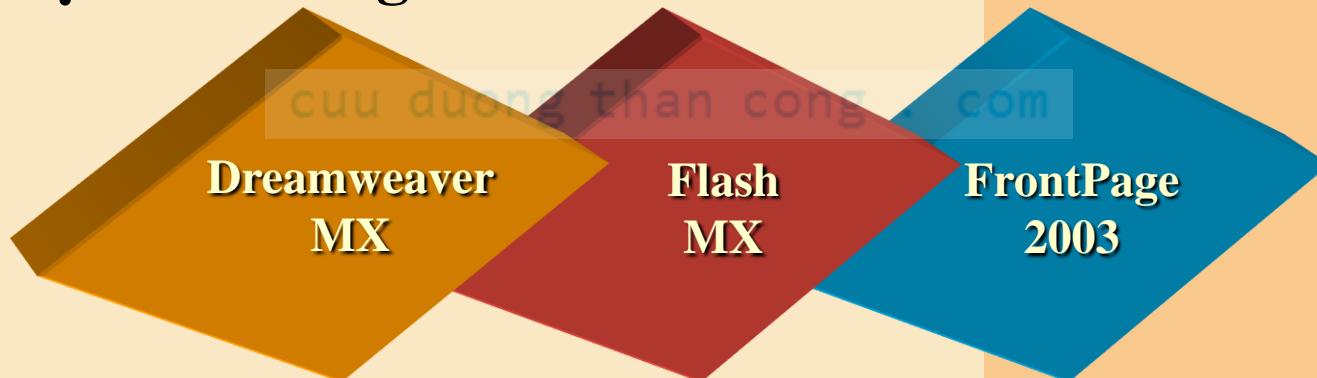
Cho phép developers có thể thiết kế những trang cho các trình duyệt chuyên dụng – mobil, ...

Sử dụng chuẩn wireless application protocol (WAP), để xác định cách thức các thiết bị không dây liên lạc với Web

Web Page Development

Web page authoring software?

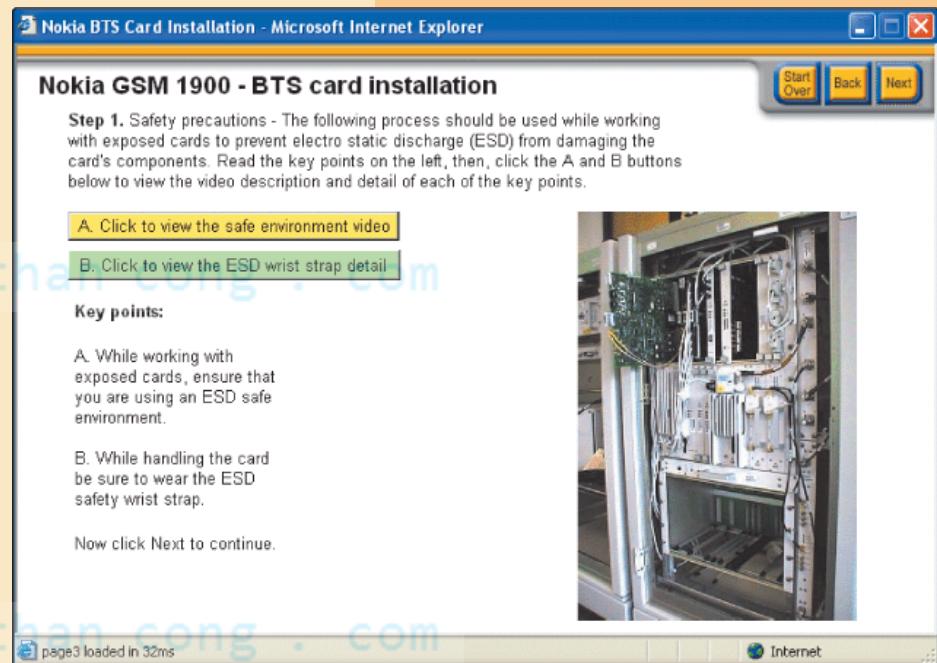
- Tạo các trang Web hoàn hảo mà không cần dùng HTML
- Tự tạo các trang HTML



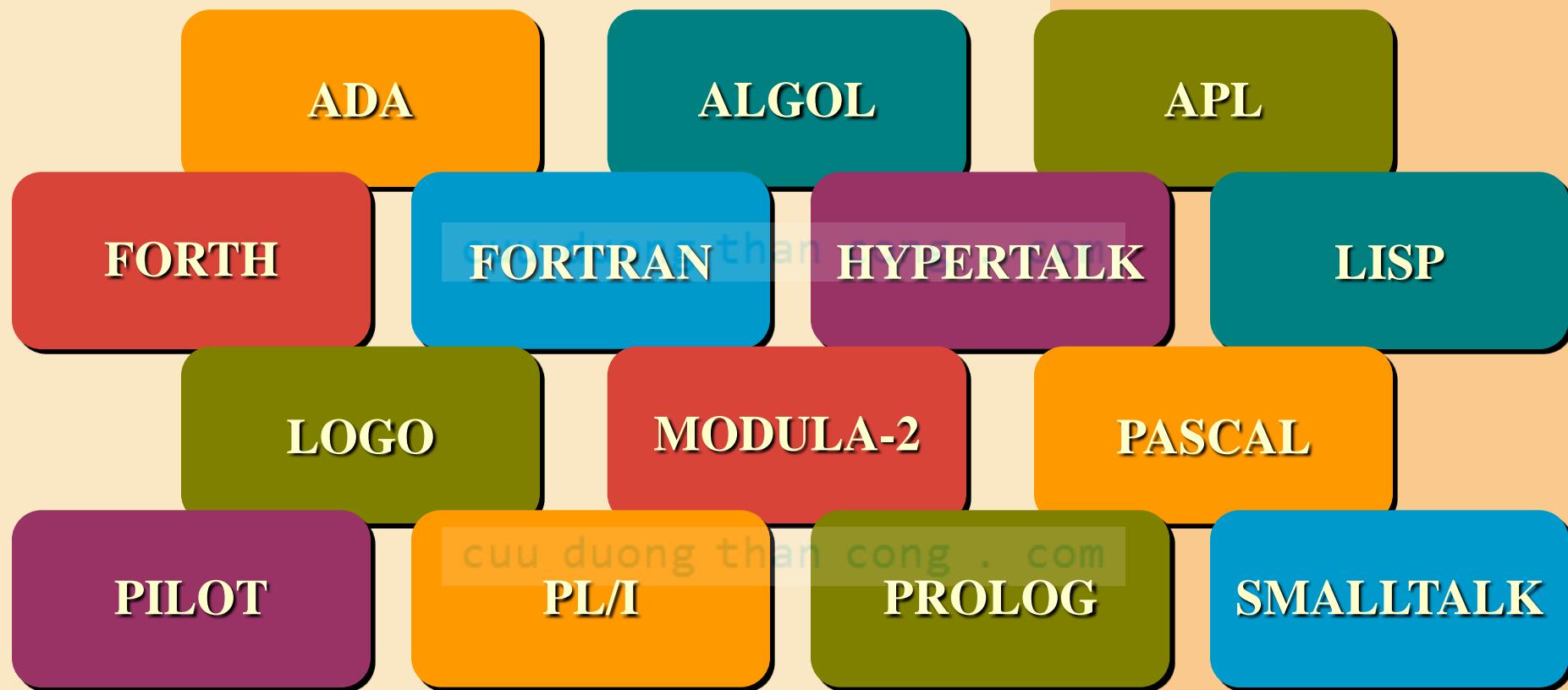
Multimedia Program Development

Multimedia authoring software?

- Kết hợp văn bản, đồ họa, hoạt hình, âm thanh và video trong 1 bài trình diễn có tương tác
- Sử dụng cho computer-based training (CBT) và Web-based training (WBT)
- Software includes Toolbook, Authorware, và Director



Các Programming Languages khác



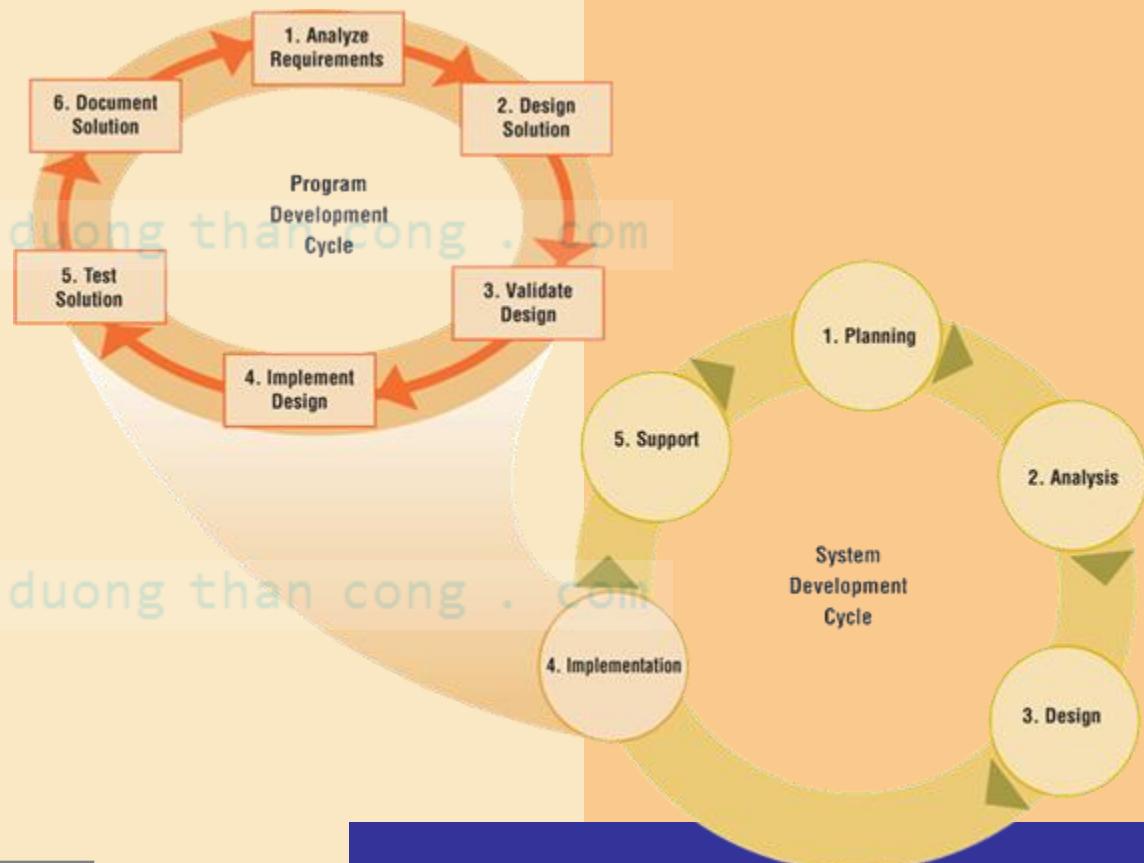
Chu trình phát triển chương trình (PM)

Program development cycle?

- Là các bước mà LTV dùng để xây dựng CT

- Programming team—

ng chương
trình



Step 1 — Analyze Requirements

Các việc cần làm khi phân tích yêu cầu?

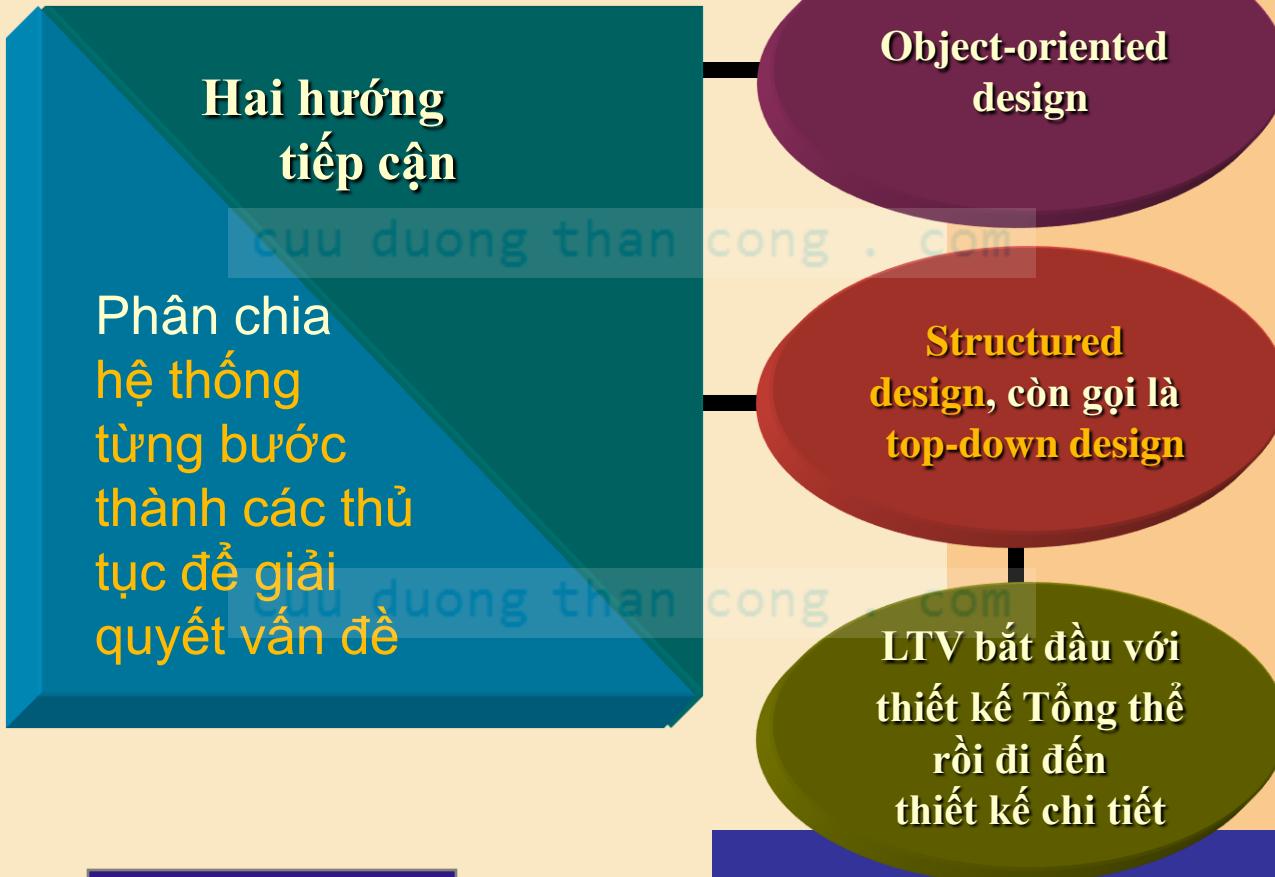
1. Khảo sát và Thiết lập các yêu cầu
2. XD các mô hình phân tích
3. Xác định đầu vào, đầu ra và các xử lý cùng các thành phần dữ liệu.

- IPO chart—Xác định đầu vào, đầu ra và các bước xử lý

IPO CHART		
Input	Processing	Output
Regular Time Hours Worked	Read regular time hours worked, overtime hours worked, hourly pay rate.	Gross Pay
Overtime Hours Worked	Calculate regular time pay.	
Hourly Pay Rate	If employee worked overtime, calculate overtime pay.	
	Calculate gross pay.	
	Print gross pay.	

Step 2 — Design Solution

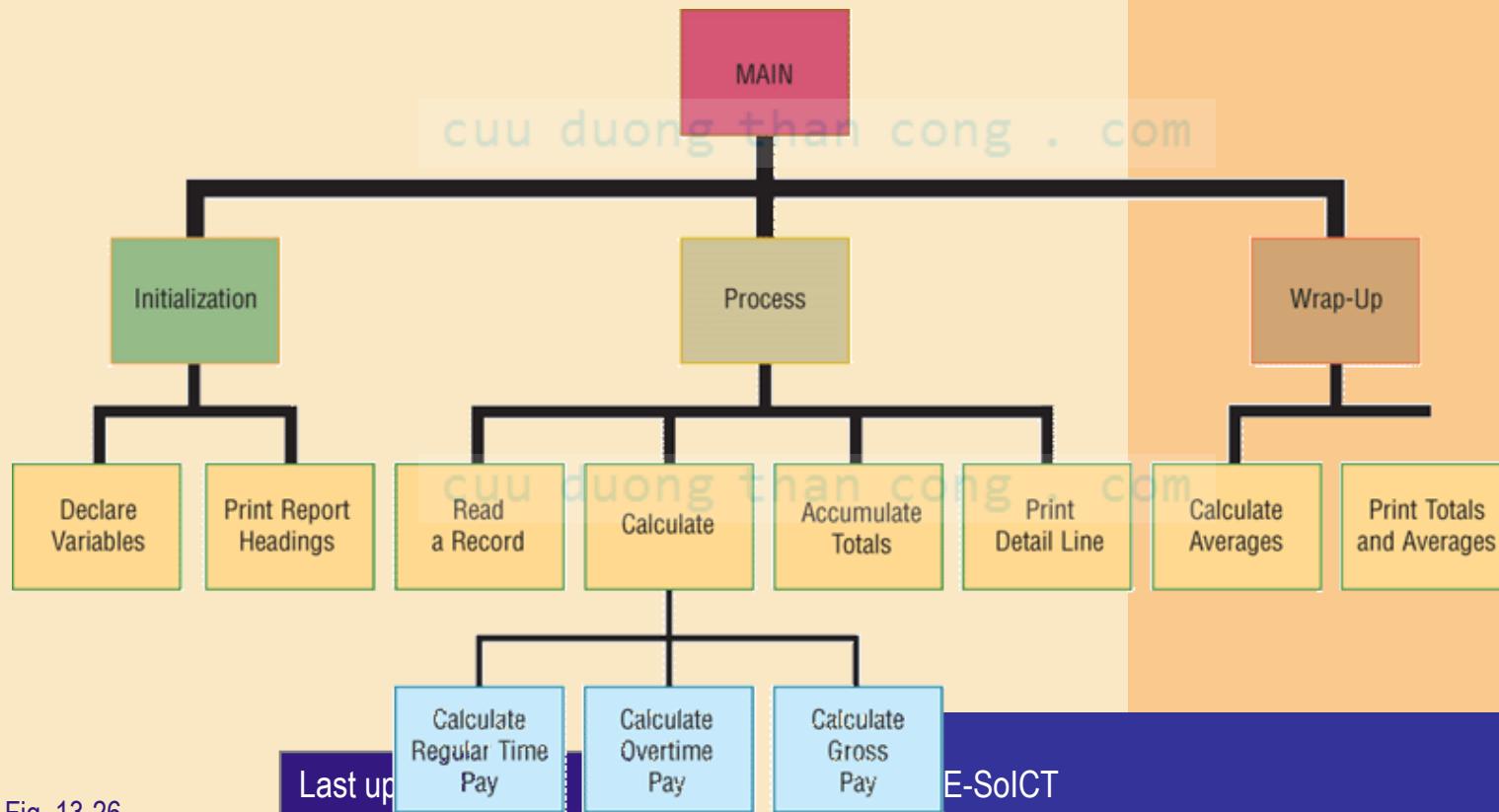
Những việc cần làm trong bước thiết kế giải pháp?



Step 2 — Design Solution

Sơ đồ phân cấp chức năng- hierarchy chart?

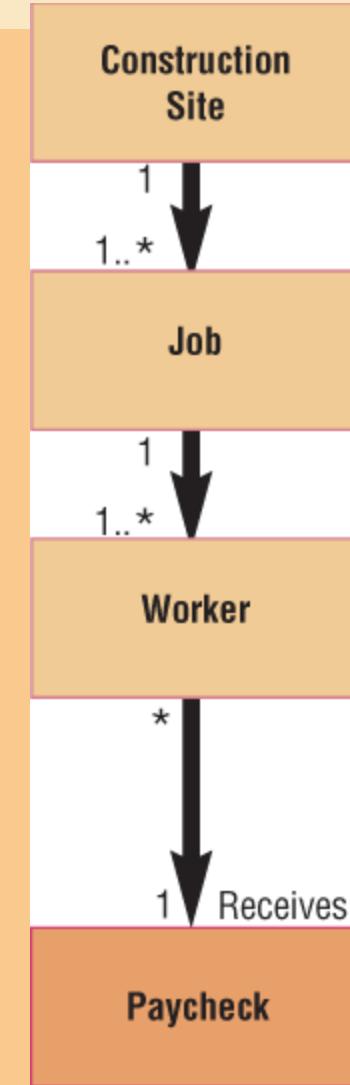
- Trực quan hóa các modules chương trình
- Còn gọi là sơ đồ cấu trúc



Step 2 — Design Solution

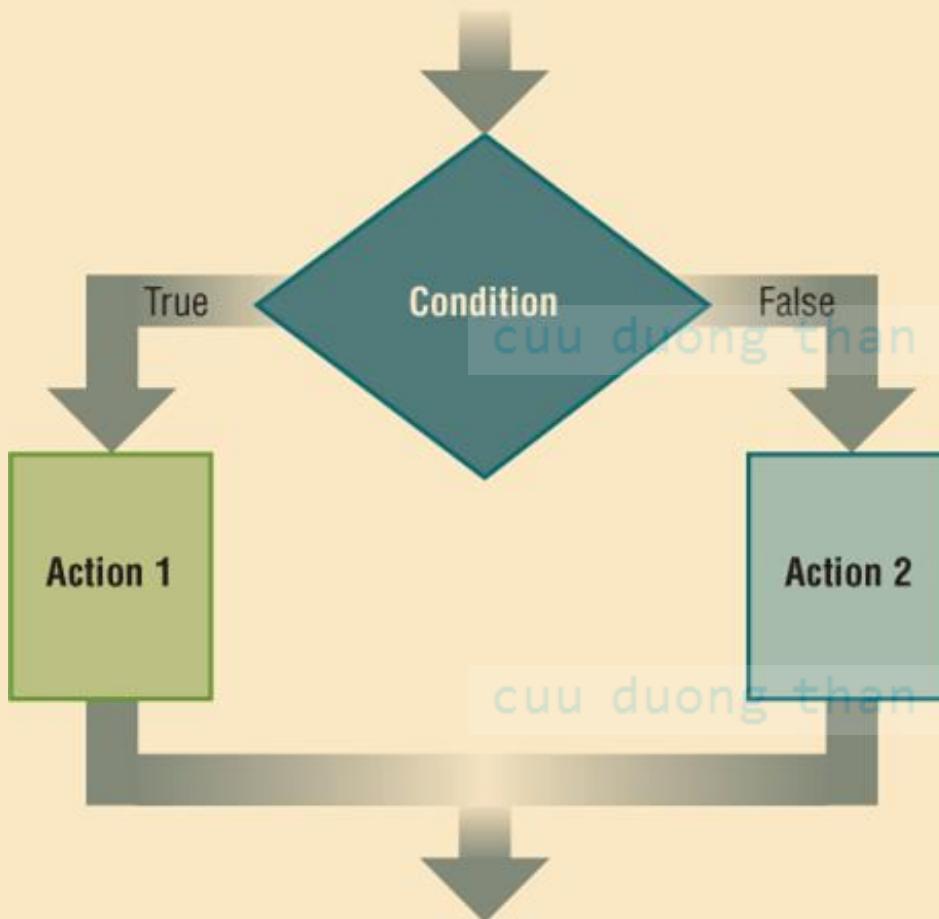
Object-oriented (OO) design là gì?

- LTV 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



Step 2 — Design Solution

Cấu trúc tuyển chọn



- Chỉ ra hành động tương ứng điều kiện 2 kiểu

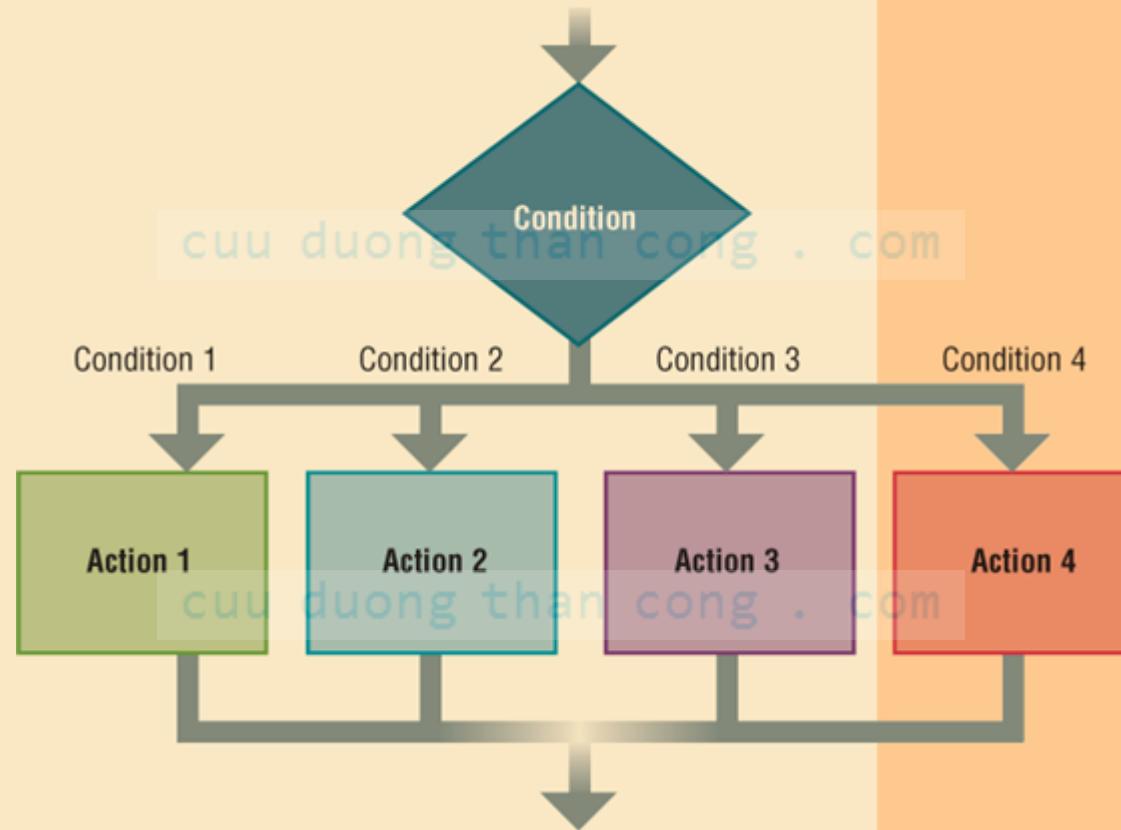
Case control structure

- **If-then-else control structure**—dựa theo 2 khả năng: true or false

Step 2 — Design Solution

Case control structure

- Dựa theo 3 hoặc nhiều hơn các khả năng

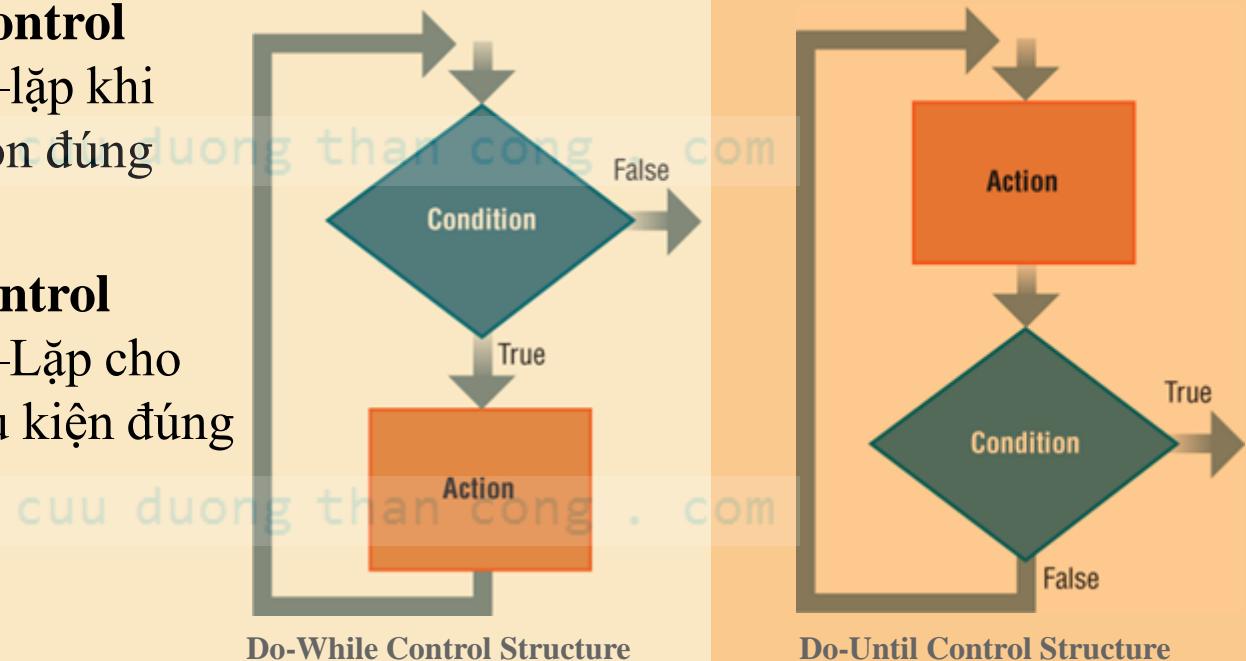


Step 2 — Design Solution

Cấu trúc lặp

- Cho phép CT thực hiện 1 hay nhiều hành động lặp đi lặp lại.

- **Do-while control structure**—lặp khi điều kiện còn đúng
- **Do-until control structure**—Lặp cho đến khi điều kiện đúng



Step 3 — Validate Design

Những điều cần làm trong giai đoạn này?

Kiểm tra
độ chính xác
của program

LTV kiểm tra
logic cho tính đúng đắn
và thử tìm các lỗi logic

Desk check
LTV dùng các dữ liệu
thử nghiệm để kiểm tra
chương trình

Logic error
các sai sót khi thiết kế
gây ra những kết quả
không chính xác

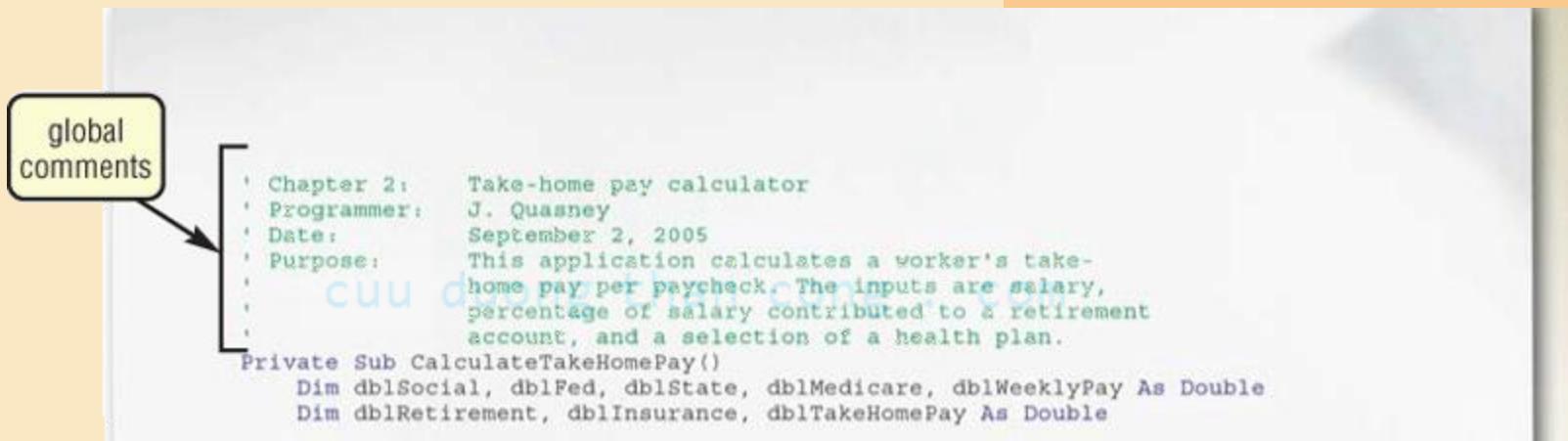
Test data
các dữ liệu thử nghiệm
giống như số liệu thực
mà CT sẽ thực hiện

Structured walkthrough
LTV mô tả logic
của thuật toán trong khi
programming team duyệt theo
logic chương trình

Step 4 — Implement Design

implementation?

- **Viết code : dịch 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**



Step 5 — Test Solution

Những việc cần làm ?

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

cuu duong than cong . com

Step 6 — Document Solution

Là bước không kém quan trọng

- 2 hoạt động

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

u

Tóm lại

Có hàng loạt các NNLT dùng để viết
computer programs

Chu trình phát triển chương trình
và các công cụ được dùng để
làm cho quá trình này hiệu quả hơn

4 Mô thức lập trình cơ bản

cuu duong than cong . com

Bàn thêm về các Mô thức lập trình

- Programming paradigm (Mô thức lập trình)
 - . Là 1 khuôn mẫu - pattern dùng như một Mô thức lập trình máy tính
 - . Là 1 Mô thức cho 1 lớp các NNLT có cùng những đặc trưng cơ bản
- Programming technique (Kỹ thuật lập trình)
 - . Liên quan đến các ý tưởng thuật toán để giải quyết một lớp vấn đề tương ứng
 - . Ví dụ: 'Divide and conquer' và 'program development by stepwise refinement'
- Programming style (Phong cách lập trình)
 - . Là cách chúng ta trình bày trong 1 computer program
 - . Phong cách tốt giúp cho chương trình dễ hiểu, dễ đọc, dễ kiểm tra -> dễ bảo trì, cập nhật, gỡ rối, tránh bị lỗi
- Programming culture (Văn hóa lập trình)
 - . Tổng hợp các hành vi lập trình, thường liên quan đến các dòng ngôn ngữ lập trình
 - . Là tổng thể của Mô thức chính, phong cách và kỹ thuật lập trình
 - . Là nhân cách đạo đức trong lập trình cũng như khai thác các CT

Bốn Mô thức lập trình cơ bản

Bốn Mô thức lập trình cơ bản là :

- Imperative paradigm
- Functional paradigm
- Logical paradigm
- Object-oriented paradigm

Ngoài ra :

- Visual paradigm
- Parallel paradigms

Một vài Mô thức mới khác :

Concurrent programming

Distributed programming

Extreme programming

...

Tuy nhiên chúng ta chỉ tập trung vào Mô thức 1 và sơ qua về 3 Mô thức cơ bản còn lại

Imperative paradigm

- ❖ Với Mô thức này ý tưởng cơ bản là các lệnh gây ảnh hưởng đáng kể đến trạng thái chương trình.
- ❖ Mỗi imperative program bao gồm:
 - *Declarative statements* – các lệnh khai báo nhằm định nghĩa các biến: tên và kiểu dữ liệu của biến. Các biến này có thể thay đổi giá trị trong quá trình thực hiện Chương trình .
cuu duong than cong . com
 - *Assignment statements* – *Lệnh gán* : gán giá trị mới cho biến
 - *Program flow control statements* – *Các lệnh điều khiển cấu trúc chương trình* : Xác định trình tự thực hiện các lệnh trong chương trình.
cuu duong than cong . com
 - Module: chia chương trình thành các chương trình con : Functions & Procedures

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 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*
- ❖ Tương ứng với cách mô tả các công việc hàng ngày như là trình tự nấu ăn hay sửa chữa xe cộ
- ❖ 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 Functional programming

```
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. Có vẻ như 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 : Tự động kiểm chứng trong trí tuệ nhân tạo
 - 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 :



ng 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

- m - 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**

Chương II

cuu duong than cong . com

trong KTLT

(6LT – 2BT)

cuu duong than cong . com

Chương II

1. cuu duong than cong . nh
c
- 2.
3. ng
4. cuu duong than cong . com
ng

c CT

•

t.

•

cuu duong than cong . com

nh con.

•

c i ND.
cuu duong than cong . com

•

C.

p)

-

ng:

— m

— c

- cuu duong than cong . com

-

cuu duong than cong . com

O.

-

ch chung .

p)

- m trong C

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

- c trong C

```
void nhapmang(Mang V, int n)
{int i;
 for (i=0;i <n;i++)
 { printf("\n V[ %d ]=%d",i);
  scanf("%d",&V[i]);
 }
}
```

p)

•

>

•

cuu duong than cong . com

•

void main(void) == main()

cuu duong than cong . com

•

C

i

- CT con.
- a CT con.

cuu duong than cong . com

int GT(int n)

c

```
{ if ( n==0) return 1;  
else return n * GT(n-1);  
}
```

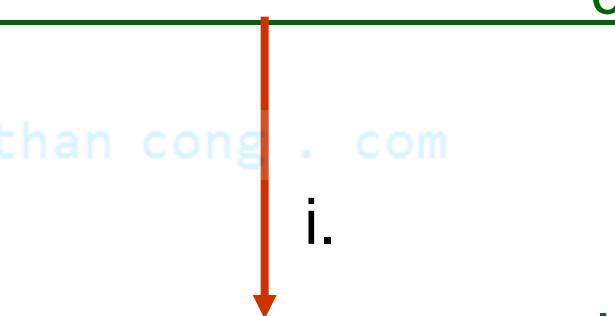
cuu duong than cong . com

....

Printf("\3!= %d",GT(3));

c

i

- c (called) “ ”
c.
-
- n trong LT).
cuu duong than cong . com
- :
- (by val): 
cuu duong than cong . com
i.
c!!!
-

p)

-

c.

i (

– side effect).

•=> C

c!!!

• TD:

• n theo tên (macro)

-

n sau).

```
#include <stdio.h>
#include <conio.h>
swap (int a, int b);
{ int temp = a;
  a = b;
  b= temp);
}
main()
{ int c = 5;
  int d = 7;
  clrscr();
  swap (c,d);
  printf("\n c= %d  d=%d",c,d);long than cong . com
  getch();
}
```

cuu duong than cong

in ra
com
c = 5 d = 7

```
#include <stdio.h>
#include <conio.h>
swap (int *a, int *b);
{ int temp = *a;
  *a = *b;
  *b= temp;
}
main()
{ int c = 5;
  int d = 7;
  clrscr();
  swap (&c,&d);
  printf("\n c= %d  d=%d",c,d);
  getch();
}
```

cuu duong than cong . long than cong . com

in ra
com
c = 7 d = 5

Tổ chức CT con

- Để tiện sử dụng CT con được tổ chức theo nhiều hình thức khác nhau:
 1. Trong cùng 1 chương trình với CT chính
 2. Ghép thành đơn vị CT
 3. Ghép thành mô đun (đơn thể chương trình)
- Cách tổ chức thứ 2 và 3 tiện dụng hơn vì tính tái sử dụng. Các CT con của ND có thể chuyển vào Thư Viện chương trình của NNLT đó (các NNLT đều có công cụ hỗ trợ việc này).
- TD: minh họa qua C/ Pascal.

Chương II

1.

nh

2.

c

3.

i

4.

ng

c (global)

- Khi 1 CT được gọi nó được nạp vào bộ nhớ và thường trú trong bộ nhớ đến khi kết thúc thực hiện. Đó chính là vòng đời của CT => Do vậy các đại lượng định nghĩa trong CT đó cũng kết thúc vòng đời của mình. => Nảy sinh khái niệm biến cục bộ và biến toàn cục.
- Biến cục bộ (local variables): Các biến được định nghĩa trong 1 CT và chỉ được sử dụng trong CT con đó. Nó có cùng vòng đời với CT sinh ra nó. Khái niệm cục bộ cũng là tương đối và phụ thuộc vào cách tổ chức CT. Nó là cục bộ của CT con đó song là toàn cục với CT con của nó.

(tiếp)

- Biến toàn cục (global variables): Các biến được định nghĩa trong 1 CT và được sử dụng trong CT con đó và các CT con của nó. => Đ/n ở 1 nơi và sử dụng ở nơi khác.

[cuuduongthancong.com](#)

- Một loại nữa là *static*: Nó là cục bộ của 1

- Chú ý cách dùng

Thí dụ

```
int V[...];  
void inmang(int n)  
{  
    int i;  
    for (i=0;i <n;i++)  
        printf(" %4d",V[i]);  
    }  
  
main ()  
{int i,j,n, tam;  
clrscr();  
printf(" So Phan tu cua mang:\n");  
scanf("%d",&n);  
nhapmang(n);  
printf("\n Day so vua nhap:");  
inmang(n);  
}
```

a CT con inmang
a main

Chương II

1.

nh

2.

c

3.

i

4.

ng

i

•

. TD:

1.

cuu duong than cong . com

ng trên.

•

cuu duong than cong . com

•

c Union

```
struct WORDREGS {  
    unsigned int ax, bx, cx, dx, si, di, cflag, flags;  
};
```

```
struct BYTEREGS {  
    unsigned char al, ah, bl, bh, cl, ch, dl, dh;  
};
```

```
union REGS {  
    struct WORDREGS x;  
    struct BYTEREGS h;  
};
```

p)

```
Typedef union {  
    struct { long abscisse ;  
            long ordonne;  
        }cart;  
    struct { cuu duong than cong . com  
            float rho;  
            float theta;  
        }pol;
```

p2
m:

P1.cart.abscisse, p2.pol.theta

p (file)

-

p.

•

cuu duong than cong . com

y).

•

c

nhau.

cuu duong than cong . com

•

phân (binary file).

phân

```
void main()
{ int i , j, n;
  mang A, B;
  FILE *fp;
  char filename[]{"Mang.Txt"};
  clrscr();
  if ((fp=fopen(filename,"w+"))==NULL)
    printf("\n Khong mo duoc tep!");
  else
  { do
    { printf("\n so phan tu (1 <n <10)");
      scanf("%d",&n);
    } while ((n < 1) ||(n>10));
    printf("\n nhap cac phan tu:");
    for (i =0;i<n;i++)
      for (j=0;j<n;j++)
        {printf("\n A[%d,%d]=",i,j);
         scanf("%d",&A[i][j]);
        }
    fwrite(&A,sizeof(int), MAX * MAX,fp);
```

phân

```
if ((fp=fopen(filename,"r+"))==NULL)
    printf("\n Khong mo duoc tep!");
else
{ fread(&B,sizeof(int), MAX * MAX,fp);
fclose(fp);
}// ket thuc else
printf("\n Mang doc ra tu tep:");
for (i = 0;i<n;i++)
{ for (j = 0;j<n;j++)    printf("\ %d ",B[i][j]);
printf("\n");
}
```

```
void main()
{ char c;
FILE *fv, *fr;
char *filename="D_ghitep.cpp";
clrscr();
if ((fv=fopen(filename,"r+"))==NULL)
    printf("\n Khong mo duoc tep!");
else
{
    filename= "D_ghitep.Txt";
    if ((fr = fopen(filename,"w+")) == NULL);
    do { c = fgetc(fv);
          fputc(c,fr);
    } while (c != EOF);
    fclose(fv);
    fclose(fr); }
```

Chương II

1. cuu duong than cong . com nh
2. cuu duong than cong . com c
3. cuu duong than cong . com i
4. cuu duong than cong . com ng

-

c). Đ c

n lén.

-

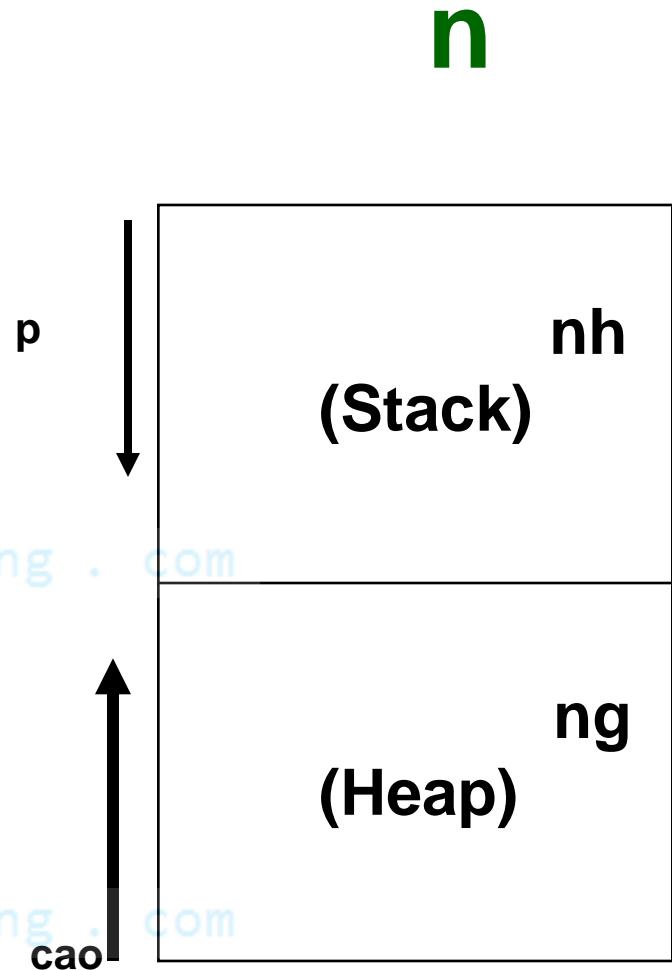
c).

cuu duong than cong . com

-

cuu duong than cong . com

Stack.



nh

-

như khai báo mảng, biến và các đối tượng 1 cách
tường minh.

-

[cuu duong than cong . com](http://cuuduongthancong.com)

nhiều đối tượng có kích
thước thay đổi linh hoạt.

-

[cuu duong than cong . com](http://cuuduongthancong.com)

n.

o trong CT).

t động

- t.
- riêng:
 - Trong C ta dùng các hàm **malloc**, **calloc**, **realloc** và **free** để xin cấp phát, tái cấp phát và giải phóng bộ nhớ. Trong C++ là **new** và **delete**.
 -
 - u).
 - o heap

cuu duong than cong . com

delete

- Để xin cấp phát bộ nhớ ta dùng :
`<biên trả> = new <kiểu dữ liệu>;`
hoặc `<biến trả> = new <kiểu dữ liệu>[số ftử];`
dòng trên xin cấp phát một vùng nhớ cho một biến đơn, còn
dòng dưới : cho một mảng các phần tử có cùng kiểu với
kiểu dữ liệu.
- Bộ nhớ động được quản lý bởi hệ điều hành, và với môi
trường đa nhiệm (multitask interface) thì bộ nhớ này sẽ
được chia sẻ giữa hàng loạt các ứng dụng, vì vậy có thể
không đủ bộ nhớ. Khi đó toán tử new sẽ trả về con trả
NULL.
- ví dụ : `int *pds;`
`pds = new int [200];`
`if (pds == NULL) { // thông báo lỗi và xử lý`

Giải phóng bộ nhớ

- delete ptr; // xóa 1 biến đơn
- delete [] ptr; // xóa 1 biến mảng
- ví dụ : #include <iostream>

```
int main() {
    int i,n; long total=100,x,*l;
    cout << "Vao so ptu ";      cin >> n;
    l = new long [n];
    if (l==NULL) exit(1);
    for (i=0;i<n;i++){
        cout <<"\n Vao so thu "<< i+1 <<":";  cin >> l[i] }
    Cout << "Danh sach cac so : \n"
    for (i=0;i<n;i++) cout << l[i] << ",";
    delete []l;
    return 0;
}
```

p 1

• n.

•

nh

-

t cuu duong than cong . com

-

-

-

n

cuu duong than cong . com

•

).

c C++.

p 2

•

ch.

•

). cuu duong than cong . com

•

C

cuu duong than cong . com

c nâng cao trong C/ C++

cuu duong than cong . com

cuu duong than cong . com

1. Mảng

- Là một dãy hữu hạn các phần tử liên tiếp có cùng kiểu và tên
- Có thể là 1 hay nhiều chiều, C không giới hạn số chiều của mảng
- Khai báo theo cú pháp sau :

DataType ArrayName [size];

hoặc **DataType ArrayName [Size1][Size2]...;**

- Khởi tạo giá trị cho mảng theo 2 cách:

- Khi khai báo :

float y[5]={3.2, 1.2, 4.5 ,6.0, 3.6}

int m[6][2] = {{1,1},{1,2},{2,1},{2,2},{3,1},{3,2}};

char s1[6] ={'H','a','n','o','i','\0'}; hoặc

char s1[6]=“Hanoi”; cuuduongthancong.com

char s1[] =“Dai hoc Bach Khoa Hanoi”; L=24

int m[][] ={{1,2,3},{4,5,6}};

- Khai báo rồi gán giá trị cho từng phần tử của mảng.

Ví dụ : int m[4];

m[0] = 1; m[1] = 2; m[2] = 3; m[3] = 4;

2. Con trỏ

- Khái niệm : Giá trị các biến được lưu trữ trong bộ nhớ MT, có thể truy cập tới các giá trị đó qua tên biến, đồng thời cũng có thể qua địa chỉ của chúng trong bộ nhớ (CTDL>).
- Con trỏ thực chất là 1 biến mà nội dung của nó là địa chỉ của 1 đối tượng khác (biến, hàm, nhưng không phải 1 hằng số).
- Có nhiều kiểu biến với các kích thước khác nhau, nên có nhiều kiểu con trỏ. Con trỏ **int** để trỏ tới biến hay hàm kiểu **int**,...
- Việc sử dụng con trỏ cho phép ta truy nhập tới 1 đối tượng gián tiếp qua địa chỉ của nó.
- Trong C, con trỏ là một công cụ rất mạnh, linh hoạt.

- Khai báo con trỏ :
- Cú pháp : dataType _* PointerName;
Chỉ rằng đây là con trỏ trả về kiểu dataType.
- Sau khi khai báo, ta được con trỏ NULL (chưa trỏ tới 1 đối tượng nào).
- Để sử dụng con trỏ, ta dùng toán tử lấy địa chỉ & PointerName = & VarName

Ví dụ :

```
int a;      int *p;      a=10;
```

```
p= &a; => *p = 10
```

- Để lấy nội dung biến do con trỏ trỏ tới, ta dùng toán tử lấy nội dung *:

* PointerName

Ví dụ :

int i,j, *p;

i = 5;

j = *p;

p = & i;

*p = j+2;

Đ/c

100		i
102		j
104		p

Gán i=5

100	5	i
102		j
104		p

gán j = *p

100	5	i
102	5	j
104	100	p

gán p = & i

100	5	i
102		j
104	100	p

*p = j+2

100	7	i
102	5	j
104	100	p

Chú ý

- Một con trỏ chỉ có thể trỏ tới 1 đối tượng cùng kiểu.
- Toán tử 1 ngôi * và & có độ ưu tiên cao hơn các toán tử số học.
- Ta có thể viết *p mọi nơi có đối tượng mà nó trỏ tới xuất hiện.

int x = 5, *p; p = &x; =>
x=x+10; ~ *p = *p+10;

- Ta cũng có thể gán nội dung 2 con trỏ cho nhau : khi đó cả hai con trỏ cùng trỏ tới 1 đối tượng.

int x=10, *p, *q;
p = &x; q = p;
=> p và q cùng trỏ tới x

Các phép toán trên con trỏ

- Một biến trỏ có thể cộng hoặc trừ với 1 số nguyên n để cho kết quả là 1 con trỏ cùng kiểu, là địa chỉ mới trỏ tới 1 đối tượng khác nằm cách đối tượng đang bị trỏ n phần tử
- Phép trừ giữa 2 con trỏ cho ta khoảng cách (số phần tử) giữa 2 con trỏ
- Không có phép cộng, nhân, chia 2 con trỏ
- Có thể dùng các phép gán, so sánh các con trỏ, nhưng cần chú ý đến sự tương thích về kiểu.

Ví dụ : char *pchar; short *pshort; long *plong;

⇒ sau khi xác lập địa chỉ cho các con trỏ, nếu :

pchar ++; pshort ++; plong ++; và các địa chỉ ban đầu tương ứng của 3 con trỏ là 100, 200 và 300, thì kết quả ta có các giá trị tương ứng là : 101, 202 và 304 tương ứng.

- Nếu viết tiếp :
plong += 5; => plong = 324 ($304 + 5 \times 4$)
pchar -= 10; => pchar = 91
pshort += 5; => pshort = 212
- Chú ý : ++ và – có độ ưu tiên cao hơn * => *p++ ~ *(p++) tức là tăng địa chỉ mà nó trỏ tới chứ không phải tăng giá trị mà nó chứa.
- *p++ = *q++ sẽ tương đương :

*p = *q;

p=p+1;

q=q+1;

Vì cả 2 phép tăng đều diễn ra sau khi phép gán được thực hiện

=> Cần dùng dấu () để tránh nhầm lẫn

Con trả void*

- Là con trả không định kiểu (**void ***). Nó có thể trả tới bất kì một loại biến nào. Thực chất một con trả void chỉ chứa một địa chỉ bộ nhớ mà không biết rằng tại địa chỉ đó có đối tượng kiểu dữ liệu gì. => không thể truy cập nội dung của một đối tượng thông qua con trả **void**. Để truy cập được đối tượng thì trước hết phải ép kiểu con trả void về con trả có định kiểu của kiểu đối tượng.

```
float x;      int y;  
void *p; // khai báo con trả void  
p = &x; // p chứa địa chỉ số thực x  
*p = 2.5; // báo lỗi vì p là con trả void  
/* cần phải ép kiểu con trả void trước khi truy cập  
đối tượng qua con trả */  
*((float*)p) = 2.5; // x = 2.5  
p = &y;           // p chứa địa chỉ số nguyên y  
*((int*)p) = 2; // y = 2
```

Con trỏ và mảng

- Giả sử ta có : int a[30]; thì & a[0] là địa chỉ phần tử đầu tiên của mảng đó, đồng thời là địa chỉ của mảng.
- Trong C, tên của mảng chính là 1 hằng địa chỉ = địa chỉ của phần tử đầu tiên của mảng.

$a = \&a[0] = a+0;$

$a+i = \&a[i];$ cuuduongthancong.com

- Tuy vậy cần chú ý rằng **a là 1 hằng** => không thể dùng nó trong câu lệnh gán hay toán tử tăng, giảm như a++;
Xét con trỏ : int *pa;

$pa = \&a[0];$

cuuduongthancong.com
=> pa trỏ vào ftử thứ nhất của mảng và :

pa +1 sẽ trỏ vào phần tử thứ 2 của mảng

$*(pa+i)$ sẽ là nội dung của a[i]

Con trỏ xâu

- Ta có : char tinhthanh[30] =“Da lat”;
- Tương đương : char *tinhthanh;
- tinhthanh=“Da lat”;
- Hoặc : char *tinhthanh =“Da lat”;
char tinhthanh[30] = “Da lat”;
- Ngoài ra các thao tác trên xâu cũng tương tự như trên mảng
- *(tinhthanh+3) = “l”
- Chú ý : với xâu thường thì không thể gán trực tiếp như dòng thứ 3

Mảng các con trỏ

Con trỏ cũng là một loại dữ liệu nên ta có thể tạo một mảng các phần tử là con trỏ theo dạng thức.

<kiểu> *<mảng con trỏ>[<số phần tử>];

- vd : char *ds[10];

⇒

của 10 xâu ký tự nào đó.

- Cũng có thể khởi tạo trực tiếp các giá trị khi khai báo
- char * ma[10] = {"mot", "hai", "ba"...};
- Chú ý : cần phân biệt mảng con trỏ và mảng nhiều chiều. Mảng nhiều chiều là mảng thực sự được khai báo và có đủ vùng nhớ dành sẵn cho các ftử. Mảng con trỏ chỉ dành không gian nhớ cho các biến trỏ (chứa địa chỉ). Khi khởi tạo hay gán giá trị : cần thêm bộ nhớ cho các ftử sử dụng => tốn nhiều hơn.

- Một ưu điểm khác của mảng trả là ta có thể hoán chuyển các đối tượng (mảng con, cấu trúc..) được trả bởi con trả này bằng cách **hoán chuyen các con trả**
- Ưu điểm tiếp theo là việc truyền tham số trong hàm
- Ví dụ : Vào ds lớp theo họ và tên, sau đó sắp xếp để in ra theo thứ tự ABC.

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

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

```
#define MAXHS 50
```

```
#define MAXLEN 30
```

```
int main () {
    int i, j, count = 0;  char ds[MAXHS][MAXLEN];
    char *ptr[MAXHS], *tmp;
    while ( count < MAXHS) {
        printf(" Hoc sinh thu : %d ",count+1);
        gets(ds[count]);
        if (strlen(ds[count]) == 0) break;
        ptr[count] = ds +count;
        ++count;
    }
    for ( i=0;i<count-1;i++)
        for ( j =i+1;j < count; j++)
            if (strcmp(ptr[i],ptr[j])>0) {
                tmp=ptr[i]; ptr[i]= ptr[j]; ptr[j] = tmp;
            }
    for (i=0;i<count; i++)
        printf("\n %d : %s", i+1,ptr[i]);
}
```

Con trỏ trỏ tới con trỏ

- Bản thân con trỏ cũng là 1 biến, vì vậy nó cũng có địa chỉ và có thể dùng 1 con trỏ khác để trỏ tới địa chỉ đó.

- <Kiểu DL> **<Tên biến trỏ>;

- Ví dụ : int x = 12;

```
int *ptr = &x;
```

```
int **ptr_to_ptr = &ptr;
```

- Có thể mô tả 1 mảng 2 chiều qua con trỏ của con trỏ theo công thức :

$$\text{ArrName}[i][k] = *(*(ArrName+i)+k)$$

Với ArrName+i là địa chỉ của phần tử thứ i của mảng

*(ArrName+i) cho nội dung ftử trên

*(ArrName+i)+k là địa chỉ phần tử [i][k]

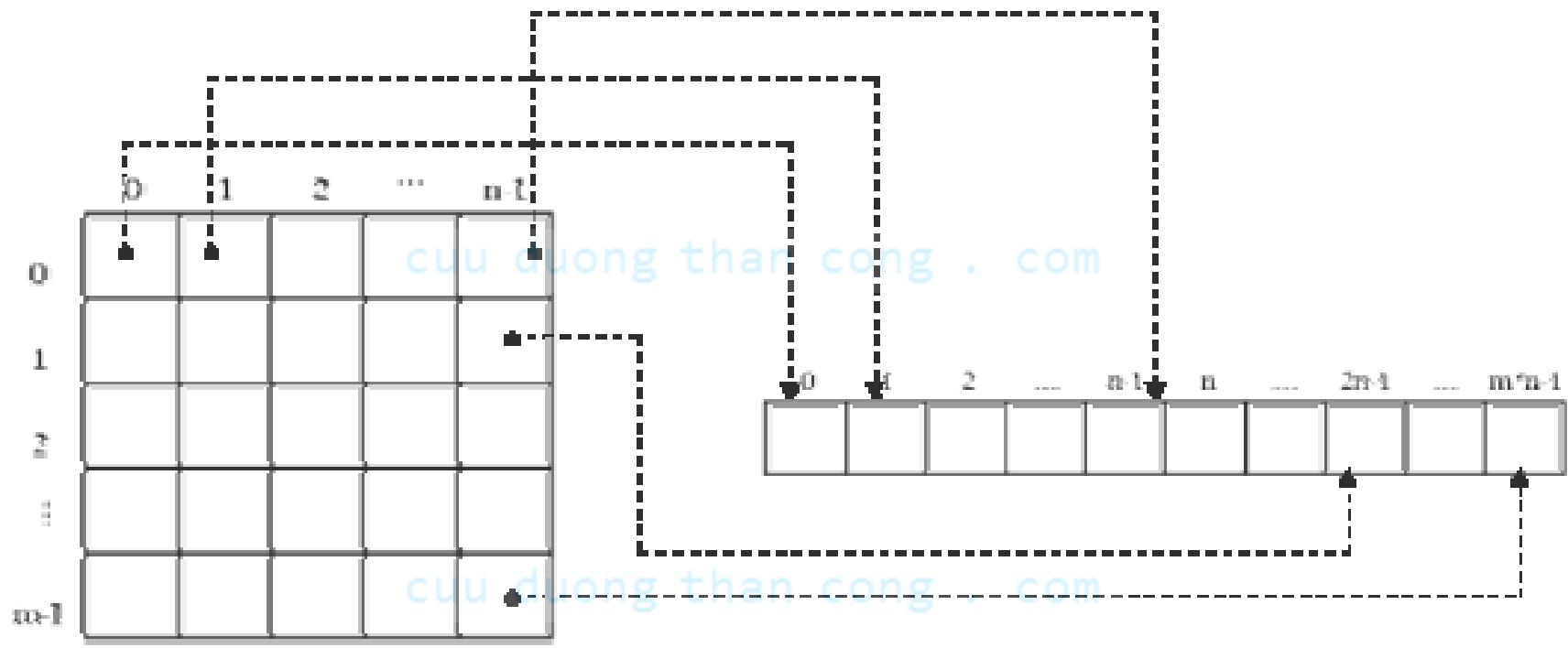
- Ví dụ : in ra 1 ma tran vuông và cộng mỗi ptử của MT với 10

```
#include <stdio.h>
#define hang 3
#define cot 3
int main() {
    int mt[hang][cot] = {{7,8,9},{10,13,15},{2,7,8}};
    int i,j;
    for (i=0; i<hang; i++) {
        for (j=0; j<cot; j++) printf(" %d ",mt[i][j]);
        printf("\n");
    }
    // cách dùng địa chỉ (con trỏ tương đương)
    for (i=0; i<hang;i++) {
        for (j=0;j<cot;j++) {

            printf(" %d ", *(*(mt+i)+j));
        }
        printf("\n");
    }
}
```

Dùng bộ nhớ động cho mảng

Ta có thể coi một mảng 2 chiều là 1 mảng 1 chiều như hình sau :



Gọi X là mảng hai chiều có kích thước m dòng và n cột.
 A là mảng một chiều tương ứng ,thì $X[i][j] = A[i*n + j]$ nếu
phân bố theo hàng.

Dùng bộ nhớ động cho mảng (cách 1)

- Với mảng số nguyên 2 chiều có kích thước là R * C ta khai báo như sau :

```
int **mt;
```

```
mt = new int *[R];
```

```
int temp = new int [R*C];
```

```
for (i=0; i< R; ++i) {
```

```
    mt[i] = temp; uu duong than cong . com
```

```
    temp += C; ? ? ?
```

```
} / Khai bao xong.
```

Sử dụng : mt[i][j] như bình thường. cuối cùng để giải phóng:

```
delete [] mt[0]; // xoá ? Tại sao?
```

```
delete [] mt;
```

p)

```
void main() {  
int *A;  
int row, col, i, j;  
clrscr();  
printf("\n so hang");  
scanf("%d",&row);  
printf("\n so cot");  
scanf("%d",&col);  
// cap phat vung nho cho mang A  
A = (int *) malloc (row * col *sizeof(int));  
// nhap mang 1  
for (i=0;i<row;i++)  
for (j=0;j<col;j++)  
{ printf("M1[%d][%d]=",i,j);  
scanf("%d",A +i*row+j);  
}  
cuu duong than cong . com
```

p)

ng

```
printf("\n mang 1\n");
for (i=0;i<row;i++)
{ for (j=0;j<col;j++) printf ("%p %d ",A +i *row+j, *(A +i
 *row+j));
printf("\n");
}
getch();
}
```

```
so hang3
cuu so cot3 than cong . com
mang 1
077C 1 077E 2 0780 3
0782 4 0784 5 0786 6
0788 7 078A 8 078C 9
```

ch 2)

```
// chuong trinh thu cap phat dong cho mang 2 chieu row x col phan tu.  
// CT cap phat lan dau theo hang: Spt bang so hang voi con tro *A.  
// CT cap phat tiep theo cot voi con tro *(A+i)  
void main() {  
    int **A;  
    int row, col, i, j;  
    clrscr();  
    printf("\n so hang");  
    scanf("%d",&row);  
    printf("\n so cot");  
    scanf("%d",&col);  
    // cap phat vung nho cho mang 1, mang 2  
    *A = (int *) malloc (row *sizeof(int));  
    for (i=0;i <row;i++)  
        *( A+i) = (int *) malloc (col *sizeof(int));
```

p)

```
// nhap mang 1
for (i=0;i<row;i++)
for (j=0;j<col;j++)
{ printf("M1[%d][%d]=",i,j);
  scanf("%d",*(A +i)+j);
}
// In mang cuu duong than cong . com
printf("\n mang 1\n");
for (i=0;i<row;i++)
{for (j=0;j<col;j++) printf (" %p %d ", *(A +i)+j, *(*(A +i) +j));
printf("\n");
}
```

CT cộng hai ma trận với mỗi ma trận được cấp phát động

```
#include <iostream.h>
#include <conio.h>
int main()
{
    int M,N;
    int *A = NULL,*B = NULL,*C = NULL;
    clrscr();
    cout<<"Nhập số dòng của ma trận:";    cin>>M;
    cout<<"Nhập số cột của ma trận:";    cin>>N;
    //Cấp phát vùng nhớ cho ma trận A
    if (!AllocMatrix(&A,M,N))
    {
        cout<<"Không có đủ bộ nhớ!"<<endl;
        return 1;
    }
    //Cấp phát vùng nhớ cho ma trận B
    if (!AllocMatrix(&B,M,N))
    {
        cout<<"Không có đủ bộ nhớ!"<<endl;
        FreeMatrix(A); //Giải phóng vùng nhớ A
        return 1;
    }
```

```
//Cấp phát vùng nhớ cho ma trận C
if (!AllocMatrix(&C,M,N))
{
    cout<<"Khong con du bo nho!"<<endl;
    FreeMatrix(A);//Giải phóng vùng nhớ A
    FreeMatrix(B);//Giải phóng vùng nhớ B
    return 1;
}
cout<<"Nhập ma trận thu 1"<<endl;
InputMatrix(A,M,N,'A');
cout<<"Nhập ma trận thu 2"<<endl;
InputMatrix(B,M,N,'B');
clrscr();
cout<<"Ma trận thu 1"<<endl;
DisplayMatrix(A,M,N);
cout<<"Ma trận thu 2"<<endl;
DisplayMatrix(B,M,N);
AddMatrix(A,B,C,M,N);
cout<<"Tổng hai ma trận"<<endl;
DisplayMatrix(C,M,N);
FreeMatrix(A);//Giải phóng vùng nhớ A
FreeMatrix(B);//Giải phóng vùng nhớ B
FreeMatrix(C);//Giải phóng vùng nhớ C
return 0;
}
```

```
//Cộng hai ma trận
void AddMatrix(int *A,int *B,int*C,int M,int N)
{
    for(int I=0;I<M*N;++I)
        C[I] = A[I] + B[I];
}
//Cấp phát vùng nhớ cho ma trận
int AllocMatrix(int **A,int M,int N)
{
    *A = new int [M*N];
    if (*A == NULL) cuu duong than cong . com
        return 0;
    return 1;
}
//Giải phóng vùng nhớ
void FreeMatrix(int *A)
{
    if (A!=NULL) cuu duong than cong . com
        delete [] A;
}
```

```

//Nhập các giá trị của ma trận
void InputMatrix(int *A,int M,int N,char Symbol)
{
    for(int I=0;I<M;++I)
        for(int J=0;J<N;++J)
    {
        cout<<Symbol<<"["<<I<<"]["<<J<<"]=";
        cin>>A[I*N+J];
    }
}
//Hiển thị ma trận
void DisplayMatrix(int *A,int M,int N)
{
    for(int I=0;I<M;++I)
    {
        for(int J=0;J<N;++J)
        {
            out.width(7);//canh le phai voi chieu dai 7 ky tu
            cout<<A[I*N+J];
        }
        cout<<endl;
    }
}

```

Phép tham chiếu

Trong C, hàm nhận tham số là con trỏ đòi hỏi chúng ta phải thận trọng khi gọi hàm. Chúng ta cần viết hàm hoán đổi giá trị giữa hai số như sau:

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

Để hoán đổi giá trị hai biến A và B thì chúng ta gọi hàm:
 $\text{Swap}(\&A, \&B);$

Rõ ràng cách viết này không được thuận tiện lắm.

Dùng tham chiếu với c++

```
void Swap(int &X, int &Y)
```

```
{  
    int Temp = X;  
    X = Y;  
    Y = Temp ;  
}
```

- Chúng ta gọi hàm như sau :
`Swap(A, B);`
- Với cách gọi ~~cú pháp~~ này, C++ tự gửi địa chỉ của A và B làm tham số cho hàm `Swap()`.

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

```
#include <iostream.h>
```

```
int X = 4;
```

```
int & MyFunc()
```

```
{
```

```
    return X;
```

```
}
```

cuu duong than cong . com

```
int main()
```

```
{
```

```
    cout<<"X="<<X<<endl;
```

```
    cout<<"X="<<MyFunc()<<endl;
```

```
    MyFunc() = 20; //Nghĩa là X = 20
```

```
    cout<<"X="<<X<<endl;
```

```
    return 0;
```

```
}
```

Phép đa năng hóa/chồng (Overloading)

- Với ngôn ngữ C++, chúng ta có thể *chồng* các hàm và các toán tử (operator). Chồng là phương pháp cung cấp nhiều hơn một định nghĩa cho tên hàm/toán tử đã cho trong cùng một phạm vi. Trình biên dịch sẽ lựa chọn phiên bản thích hợp của hàm hay toán tử dựa trên các tham số mà nó được gọi.
- Với C, tên hàm phải là duy nhất

Chồng hàm (Functions overloading)

- Trong c phải dùng 3 hàm để tính trị tuyệt đối ? :
int abs(int i);
long labs(long l);
double fabs(double d);
- C++ cho phép chúng ta tạo ra các hàm khác nhau có cùng một tên.
int abs(int i);
long abs(long l);
double abs(double d);

```
#include <iostream.h>
#include <math.h>
int MyAbs(int X) {
    return abs(X);
}
long MyAbs(long X) {
    return labs(X);
}
double MyAbs(double X) {
    return fabs(X);
}
int main() {  
    int X = -7;  
    long Y = 200000L;  
    double Z = -35.678;  
    cout<<"Tri tuyet doi cua so nguyen "<<X<<" la "  
<<MyAbs(X)<<endl;  
    cout<<"Tri tuyet doi cua so nguyen "<<Y<<" la "  
<<MyAbs(Y)<<endl;  
    cout<<"Tri tuyet doi cua so thuc "<<Z<<" la " <<MyAbs(Z)<<endl;  
    return 0;  
}
```

Chồng toán tử

- Trong ngôn ngữ C, khi chúng ta tự tạo ra một kiểu dữ liệu mới, chúng ta thực hiện các thao tác liên quan đến kiểu dữ liệu đó thường thông qua các hàm, điều này trở nên không thoải mái.
- Ví dụ : cài đặt các phép toán cộng và trừ số phức

```
#include <stdio.h> /* Dinh nghia so phuc */
struct SP {
    double THUC;           double Image; } ;
SP SetSP(double R,double I);
SP AddSP(SP C1,SP C2);
SP SubSP(SP C1,SP C2);
void DisplaySP(SP C);
int main(void) {
    SP C1,C2,C3,C4;
    C1 = SetSP(1.0,2.0);  
CuuDuongThanCong . com
    C2 = SetSP(-3.0,4.0);
    cout << "\nSo phuc thu nhat:";           DisplaySP(C1);
    cout << "\nSo phuc thu hai:";             DisplaySP(C2);
    C3 = AddSP(C1,C2);
    C4 = SubSP(C1,C2);
    cout << "\nTong hai so phuc nay:";       DisplaySP(C3);
    cout << "\nHieu hai so phuc nay:";        DisplaySP(C4);
    return 0;
}
```

```
SP SetSP(double R,double I) {  
    SP Tmp;  
    Tmp.THUC = R; Tmp.Image = I;  
    return Tmp; }  
SP AddSP(SP C1,SP C2) {  
    SP Tmp;  
    Tmp.THUC = C1.THUC+C2.THUC;  
    Tmp.Image = C1.Image+C2.Image;  
    return Tmp; }  
SP SubSP(SP C1,SP C2) {  
    SP Tmp;  
    Tmp.THUC = C1.THUC-C2.THUC;  
    Tmp.Image = C1.Image-C2.Image;  
    return Tmp; }  
void DisplaySP(SP C) { cout <<C.THUC <<' i ' <<C.Image;  
}  
Last update 8-2010  
CuuDuongThanCong.com
```

C++

- Trong ví dụ trên, ta dùng hàm để cài đặt các phép toán cộng và trừ hai số phức ; => phức tạp, không thoải mái khi sử dụng, vì thực chất thao tác cộng và trừ là các toán tử chứ không phải là hàm.
- C++ cho phép chúng ta có thể định nghĩa lại chức năng của các toán tử đã có sẵn một cách tiện lợi và tự nhiên hơn rất nhiều. Điều này gọi là chèn toán tử.
- Một hàm định nghĩa một toán tử có cú pháp sau:
data_type operator operator_symbol (parameters)

```
{ .....  
}
```

Trong đó:

- data_type*: Kiểu trả về.
- operator_symbol*: Ký hiệu của toán tử.
- parameters*: Các tham số (nếu có).

```
#include <iostream.h> //Dinh nghia so phuc
struct { double THUC; double AO; }SP;
SP SetSP(double R,double I);
void DisplaySP(SP C);
SP operator + (SP C1,SP C2);
SP operator - (SP C1,SP C2);
int main() {
    SP C1,C2,C3,C4;      C1 = SetSP(1.1,2.0);
    C2 = SetSP(-3.0,4.0); cout<<"\nSo phuc thu nhat:";
    DisplaySP(C1);        cout<<"\nSo phuc thu hai:";
    DisplaySP(C2);        C3 = C1 + C2;
    C4 = C1 - C2;         cout<<"\nTong hai so phuc nay:";
    DisplaySP(C3);        cout<<"\nHieu hai so phuc nay:";
    DisplaySP(C4);
    return 0; }
```

```
SetSP(double R,double I) {  
    SP Tmp;  
    Tmp.THUC = R; Tmp.AO = I; return Tmp; }  
    //Cong hai so phuc  
SP operator + (SP C1,SP C2) { SP Tmp;  
Tmp.THUC = C1.THUC+C2.THUC;  
Tmp.AO = C1.AO+C2.AO; return Tmp; }  
//Tru hai so phuc  
SP operator - (SP C1,SP C2) { SP Tmp;  
Tmp.THUC = C1.THUC-C2.THUC;  
Tmp.AO = C1.AO-C2.AO; return Tmp; }  
//Hien thi so phuc  
void DisplaySP(SP C) {  
    cout<<"("<<C.THUC<<","<<C.AO<<")"; }
```

Các giới hạn của chèn toán tử

- Không thể định nghĩa các toán tử mới.
- Hầu hết các toán tử của C++ đều có thể được chèn. Các toán tử sau không được chèn là :
 - :: Toán tử định phạm vi.
 - .* Truy cập đến con trỏ là trường của **struct** hay **class**.
 - . Truy cập đến trường của **struct** hay **class**.
 - ? Toán tử điều kiện

sizeof

Các ký hiệu tiền xử lý .

- Không thể thay đổi thứ tự ưu tiên của một toán tử cũng như số các toán hạng của nó.
- Không thể thay đổi ý nghĩa của các toán tử khi áp dụng cho các kiểu có sẵn.
- Chèn các toán tử không thể có các tham số có giá trị mặc định.

cuu duong than cong . com

(3LT – 2BT)

cuu duong than cong . com

Efficient Programs

- Trước hết là giải thuật
 - Hãy dùng giải thuật hay nhất có thể
 - Sau đó hãy nghĩ tới việc tăng tính hiệu quả của code
 - Ví dụ : Tính tổng của n số tự nhiên kể từ m

```
void main() {  
    long n,m,i , sum ;  
    cout << ' vào n ' ; cin << n;  
    cout << ' vào m ' ; cin << m;  
    sum =0;  
    for(i = m ; i < m+n; i++) {  
        sum += i;  
    }  
    cout << ' Tổng = ' <<sum;  
}
```

```
void main()  
{  
    long n,m , sum ;  
    cout << ' vào n ' ; cin << n;  
    cout << ' vào m ' ; cin << m;  
    sum =(m + m+ n-1) * n / 2;  
    cout << ' Tổng = ' <<sum;  
}  
//TD m=3, n=4 => KQ = 18!
```

Dùng chỉ thị chương trình dịch

- Một số Chương trình dịch có vai trò rất lớn trong việc tối ưu chương trình.
 - Chúng phân tích sâu mã nguồn và làm mọi điều “machinely” có thể.
 - Ví dụ GNU g++ compiler trên Linux/Cygwin cho chương trình viết bằng c:
 - g++ **-O5** –o myprog myprog.ccó thể cải thiện hiệu năng từ 10% đến 300%

Nhưng...

- Bạn vẫn có thể thực hiện những cải tiến mà trình dịch không thể.
- Bạn phải loại bỏ tất cả những chẽ bất hợp lý trong code:
 - Làm cho chương trình hiệu quả nhất có thể
- Có thể phải xem lại khi thấy chương trình chạy chậm.

Vậy cần tập trung vào đâu để cải tiến nhanh nhất, tốt nhất ?

Writing Efficient Code

- Xác định nguồn gây kém hiệu quả:
 - Dư thừa tính toán - redundant computation
 - Chủ yếu cuuduongthancong.com
 - Trong các procedures
 - Các vòng lặp : Loops

cuuduongthancong.com

Khởi tạo 1 lần, dùng nhiều lần

- Before

```
float f()
{ double value = sin(0.25);
//
....
```

cuu duong than cong . com

- After

```
double defaultValue = sin(0.25);

float f()
{ double value = defaultValue;
//
....
```

cuu duong than cong . com

Inline functions

- Nếu 1 hàm trong c++ chỉ gồm những lệnh đơn giản, không có 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 nhanh hơn , không dùng stack – 4 bước khi 1 hàm được gọi ...

cuu duong than cong . com

Inline functions

```
#include <iostream>
#include <cmath>
using namespace std;
inline double hypotenuse (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 << hypotenuse (k, m) << endl;
    cout << sqrt (k * k + m * m) << endl;
    return 0;
}
```

Static Variables

- Kiểu dữ liệu Static tham chiếu tới **global** hay '**static variables**' , chúng được cấp phát bộ nhớ khi dịch compile-time.

```
int int_array[100];
int main() {
    static float float_array[100];
    double double_array[100];
    char *pchar;
    pchar = (char *)malloc(100);
    /* .... */
    return (0);
}
```

Static Variables

- Các biến khai báo trong CT con được cấp phát bộ nhớ khi CT con được gọi và sẽ được giải phóng khi CT con kết thúc.
- Khi gọi lại CT con, các biến cục bộ lại được cấp phát và khởi tạo lại, ...
- Nếu muốn 1 giá trị vẫn **được lưu lại cho đến khi kết thúc toàn chương trình**, cần khai báo biến cục bộ của CT con đó là **static** và khởi tạo cho nó 1 giá trị.
– Việc khởi tạo sẽ chỉ thực hiện lần đầu tiên chương trình được gọi và giá trị sau khi biến đổi sẽ được lưu cho các lần gọi sau.
– Bằng cách này một ct con có thể “nhớ” một vài mẩu tin sau mỗi lần được gọi.
- Dùng biến Static thay vì Global :
– Cái hay của một biến static là nó là **của CT con**, => tránh được các side efects.

Macros

—
#define max(a,b) (a>b?a:b)

- Các hàm Inline cũng giống như macros vì cả 2 được khai triển khi dịch (compile time)
 - macros được khai triển bởi preprocessor, còn inline functions được truyền bởi compiler.
- Tuy nhiên có nhiều điểm khác biệt:
 - Inline functions tuân thủ các thủ tục như 1 hàm bình thường.
 - p như các hàm khác, song có thêm từ khóa **inline** khi khai báo hàm.
 - Các biểu thức truyền như là đối số cho inline functions được tính 1 lần. Trong 1 số trường hợp, biểu thức truyền như tham số cho macros có thể được tính lại nhiều hơn 1 lần.
 - Bạn không thể gỡ rối cho macros, nhưng với inline functions thì có thể.

Tính toán trước các giá trị

- Nếu bạn phải tính đi tính lại 1 biểu thức, thì nên tính trước 1 lần và lưu lại giá trị, rồi dùng giá trị ấy sau này

```
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; }
```

Loại bỏ những biểu thức “thông thường”

- Đừng tính cùng một biểu thức nhiều lần!
- Một số compilers có thể nhận biết và xử lý.

```
for (i = 1; i<=10;i++) x += strlen(str);  
Y = 15 + strlen(str);
```

```
len = strlen(str);  
for (i = 1;i<=10;i++) x += len;  
Y = 15 + len;
```

Sử dụng các biến đổi số học!

- Trình dịch không thể tự động xử lý

```
if (a > sqrt(b))
```

```
x = a*a + 3*a + 2;
```

p nhân hơn!



```
if (a *a > b)
```

```
x = (a+1)*(a+2);
```

Dùng “lính canh” -Tránh những kiểm tra không cần thiết

- Trước

```
char s[100], searchValue;  
int pos,tim, size ;  
....  
..... Gán giá trị cho s, searchValue  
...  
size = strlen(s);  
pos = 0;  
while (pos < size) && (s[pos] != searchValue)  
    do pos++;  
If (pos >= size) tim =0 else  
tim = 1;
```

Dùng “lính canh”

- Ý tưởng chung
 - Đặt giá trị cần tìm vào cuối xâu: “ *nh canh*”
 - Luôn tìm thấy !
 - cuu duong than cong . com m thấy !

```
size = strlen(s);
strcat(s, searchValue);
pos = 0;
while ( s[pos] != searchValue)
    do pos++;
If (pos >= size) tim =0 else
    tim = 1;
```

Có thể làm tương tự với mảng, danh sách ...

Dịch chuyển những biểu thức bất biến ra khỏi vòng lặp

- Đừng lặp các biểu thức tính toán không cần thiết
- Một số Compilers có thể tự xử lý!

```
for (i =0; i<100;i++)  
    plot(i, i*sin(d));
```

```
M = sin(d);  
for (i =0; i<100;i++)  
    plot(i, i*M);
```

Không dùng các vòng lặp ngắn

```
for (i =j; i<= j+3;i++)  
    sum += q*i -i*7;
```

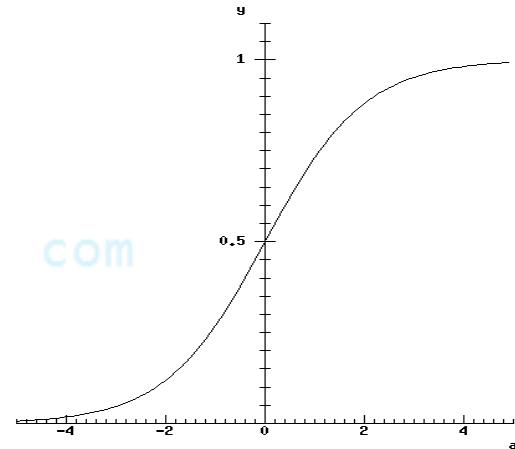
cuu duong than cong . com

```
i = j;  
sum += q*i -i*7;  
i ++;  
sum += q*i -i*7;  
i ++;  
sum += q*i-i*7;
```

Giảm thời gian tính toán

- Trong mô phỏng Neural Network người ta thường dùng hàm có tên **sigmoid**
- Với X dương lớn ta có $\text{sigmoid}(x) \approx 1$
- Với x âm “lớn”
 $\text{sigmoid}(x) \approx 0$

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-kx}}$$



Tính Sigmoid

```
float sigmoid (float x )  
{  
    return 1.0 / (1.0 + exp(-x))  
};
```

cuu duong than cong . com

Tính Sigmoid

- Hàm $\exp(-x)$ mất rất nhiều thời gian để tính!
 - Những hàm kiểu này người ta phải dùng khai triển chuỗi
 - Chuỗi Taylor /Maclaurin
 - Tính tổng các số hạng dạng $((-x)^n / n!)$
 - Mỗi số hạng lại dùng các phép toán với số chấm động
- Nói chung các mô phỏng neural network gọi hàm này **trăm triệu lần** trong mỗi lần thực hiện.
- Chính vì vậy , sigmoid(x) chiếm phần lớn thời gian (khoảng 70-80%)

Tính Sigmoid – Giải pháp

- Thay vì tính hàm mọi lúc, người ta:
 - Tính hàm tại N điểm và xây dựng 1 mảng. [cuu duong than cong . com](http://cuuduongthancong.com)
 - Trong mỗi lần gọi sigmoid
 - Tìm giá trị gần nhất của x và kết quả ứng với giá trị ấy
 - Thực hiện nội suy tuyến tính - linear interpolation

$x_0 \text{ sigmoid}(x_0)$

$x_1 \text{ sigmoid}(x_0)$

$x_2 \text{ sigmoid}(x_0)$

$x_3 \text{ sigmoid}(x_0)$

$x_4 \text{ sigmoid}(x_0)$

$x_5 \text{ sigmoid}(x_0)$

$x_6 \text{ sigmoid}(x_0)$

⋮

⋮

$x_{99} \text{ sigmoid}(x_{99})$

Tính Sigmoid (tiếp)

```
    ←
```

```
if (x < x0) return (0.0);
```

$x_0 \text{ sigmoid}(x_0)$

$x_1 \text{ sigmoid}(x_0)$

$x_2 \text{ sigmoid}(x_0)$

$x_3 \text{ sigmoid}(x_0)$

$x_4 \text{ sigmoid}(x_0)$

$x_5 \text{ sigmoid}(x_0)$

$x_6 \text{ sigmoid}(x_0)$

.

.

.

cuu duong than cong . com

$x_{99} \text{ sigmoid}(x_{99})$

```
    ←
```

```
if (x > x99) return (1.0);
```

Tính Sigmoid (tiếp)

- Chọn số các điểm ($N = 1000, 10000$, v.v.) tùy theo độ chính xác mà bạn muốn
 - Tốn kém thêm không gian bộ nhớ cho mỗi điểm là 2 float hay double tức là 8 – 16 bytes/điểm
- Khởi tạo giá trị cho mảng khi bắt đầu thực hiện.

cuu duong than cong . com

Tính Sigmoid (tiếp)

- Bạn đã biết X_o
 - Tính Delta = $X_1 - X_o$
 - Tính $X_{max} = X_o + N * \Delta$;
- Với X đã cho
 - Tính $i = (X - X_o) / \Delta$;
 - 1 phép trừ số thực và 1 phép chia số thực
 - Tính sigmoid(x)
 - 1 phép nhân float và 1 phép cộng float

Kết quả

- Nếu dùng $\exp(x)$:
 - Mỗi lần gọi mất khoảng **300 nanoseconds** với 1 máy Pentium 4 tốc độ 2 Ghz.
- Dùng tìm kiếm trên mảng và nội suy tuyến tính :
 - Mỗi lần gọi mất khoảng **30 nanoseconds**
- Tốc độ tăng gấp 10 lần
 - Đổi lại phải tốn kém thêm từ 64K -> 640 K bộ nhớ.

Lưu ý !

- Với phần lớn các chương trình, việc tăng tốc độ thực hiện là cần thiết
- Tuy nhiên, cố tăng tốc độ cho những đoạn code không sử dụng thường xuyên là vô ích !

cuu duong than cong . com

Những quy tắc cơ bản Fundamental Rules

- 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 n 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 trường , mỗi thuộc tính trong cấu trúc dữ liệu.
- Liên kết sớm - Early Binding:
 - Hãy thực hiện ngay công việc để tránh thực hiện nhiều lần sau này.

Quy tắc tăng tốc độ

Có thể tăng tốc độ bằng cách sử dụng thêm bộ nhớ (mảng).

- **Dùng thêm các dữ liệu có cấu trúc:**
 - Thời gian cho các phép toán thông dụng có thể giảm bằng cách sử dụng thêm các cấu trúc dữ liệu với các dữ liệu bổ sung hoặc bằng cách thay đổi các dữ liệu trong cấu trúc sao cho dễ tiếp cận hơn.
- **Lưu các kết quả được tính trước:**
 - Thời gian tính toán lại các hàm có thể giảm bớt bằng cách tính toán hàm chỉ 1 lần và lưu kết quả, những yêu cầu sau này sẽ được xử lý bằng cách tìm kiếm từ mảng hay danh sách kết quả thay vì tính lại hàm.
- ...

Quy tắc tăng tốc độ : cont.

- **Caching:**

- Dữ liệu thường dùng cần phải dễ tiếp cận nhất, luôn hiện hữu.

- **Lazy Evaluation:**

- Không bao giờ tính 1 phần tử cho đến khi cần để tránh những sự tính toán không cần thiết.

Quy tắc lặp : Loop Rules

Những điểm nóng - Hot spots trong phần lớn các chương trình đến từ các vòng lặp:

- **Đưa Code ra khỏi các vòng lặp:**
 - Thay vì thực hiện việc tính toán trong mỗi lần lặp, tốt nhất thực hiện nó chỉ một lần bên ngoài vòng lặp- nếu được.
- **Kết hợp các vòng lặp – loop fusion:**
 - Nếu 2 vòng lặp gần nhau cùng thao tác trên cùng 1 tập hợp các phần tử thì cần kết hợp chung vào 1 vòng lặp.

Quy tắc lặp : Loop Rules

- **Kết hợp các phép thử - Combining Tests:**
 - Trong vòng lặp càng ít kiểm tra càng tốt và tốt nhất chỉ một phép thử. LTV có thể phải thay đổi điều kiện kết thúc vòng lặp. “Lính canh (sentinel)” là một ví dụ cho quy tắc này.
- **Loại bỏ Loop :**
 - Với những vòng lặp ngắn thì cần loại bỏ vòng lặp, tránh phải thay đổi và kiểm tra điều kiện lặp.

Procedure Rules

- Khai báo những hàm ngắn và đơn giản (thường chỉ 1 dòng) là inline
 - Tránh phải thực hiện 4 bước khi hàm được gọi
 - Tránh dùng bộ nhớ stack

GOOD PROGRAMMING STYLE

Plauger. 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. **Nói chung sẽ là tốt nếu ta tuân thủ các quy tắc sau đây :**

- “Tính nhất quán”. Nếu bạn chấp nhận một cách thức đặt tên hàm hay biến, hằng thì hãy tuân thủ nó trong toàn bộ chương trình.
- Đầu mỗi CT, nên có một đoạn chú thích ...
- Mỗi CT con phải có một nhiệm vụ rõ ràng. Một CT con phải đủ ngắn để người đọc có thể nắm bắt như một đơn vị, 1 chức năng.
- Hãy dùng tối thiểu số các tham số của CT con. > 6 tham số cho 1 CT con là quá nhiều.

GOOD PROGRAMMING STYLE

- Có 2 loại Ct con : functions và procedures. Functions chỉ nên tác động tối duy nhất 1 giá trị - giá trị trả về của hàm.
- Không nên thay đổi giá trị của biến chạy trong thân của vòng lặp for, ví dụ không nên làm như sau :

```
for (i=1; i<=10; i++) i++;
```
- có cùng tên. Nếu "i" được dùng làm biến chạy cho vòng lặp trong 1 CT con, thì đừng dùng nó cho việc khác trong các CT con khác.

GOOD PROGRAMMING STYLE

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* – *Trình bày vấn đề 1 cách đơn giản, trực tiếp* cuduongthancong . com
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* cuduongthancong . com

ng cho hiệu quả

GOOD PROGRAMMING STYLE

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
8. *Parenthesize to avoid ambiguity.* – Dùng () 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 các 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

GOOD PROGRAMMING STYLE

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. – Mô đưl hóa.
Dùng các hàm và thủ tục
15. Avoid gotos completely if you can keep the program readable. – Tránh hoàn toàn việc dùng goto
16. Don't patch bad code /{ rewrite it. – Không chắp 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

GOOD PROGRAMMING STYLE

18. *Use recursive procedures for recursively-defined data structures.* –
Hãy dùng các thủ tục đệ quy cho các cấu trúc dữ liệu đệ quy.
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
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

GOOD PROGRAMMING STYLE

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
26. Check some answers by hand. – Kiểm tra 1 số câu trả lời bằng tay
27. 10.0 times 0.1 is hardly ever 1.0 . – 10 nhân 0.1 không chắc đã = 1.0
28. $7/8$ is zero while $7.0/8.0$ is not zero. $7/8 = 0$ nhưng $7.0/8.0 <> 0$?
29. Make it right before you make it faster. – Hãy làm cho CT chạy đúng, trước khi làm nó chạy nhanh

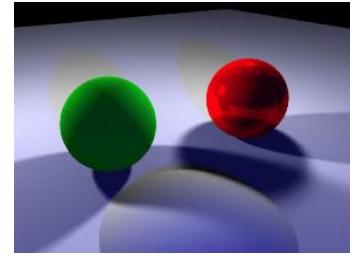
cuu duong than cong . com

GOOD PROGRAMMING STYLE

30. *Make it clear before you make it faster.* – Hãy viết code rõ ràng, trước khi làm nó chạy nhanh
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.* – Hãy dùng các tên biến có nghĩa
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

Program Style

- Who reads your code?
 - The compiler
 - Other programmers



cuu duong than cong . com

cuu duong than cong . com

Program Style

- Vì sao program style lại quan trọng?
 - Lỗi thường xảy ra do sự nhầm lẫn của LTV
 - Biến này được dùng làm gì?
 - Hàm này được gọi như thế nào?
 - Good code = code dễ đọc
- Làm thế nào để code thành dễ đọc?
 - Cấu trúc chương trình rõ ràng, dễ hiểu, khúc triết
 - Sử dụng thành ngữ phổ biến - idiom
 - Chọn tên phù hợp, gợi nhớ
 - Viết chú thích rõ ràng
 - Sử dụng môđul

Structure: Spacing

- Use readable/consistent spacing

- VD: Gán mỗi phần tử mảng $a[j] = j$.

- Bad code

```
for (j=0; j<100; j++) a[j]=j;
```

cuuduongthancong . com

- Good code

```
for (j=0; j<100; j++)
    a[j] = j;
```

cuuduongthancong . com

- Thường có thể dựa vào auto-indenting, tính năng trong trình soạn thảo

Structure: Indentation (cont.)

- Use readable/consistent indentation

-VD:

```
if (month == FEB) {  
    if (year % 4 == 0)  
        if (day > 29)  
            legal = FALSE;  
    else  
        if (day > 28)  
            legal = FALSE;  
}
```

Wrong code
(else của “if day > 29”)

```
if (month == FEB) {  
    if (year % 4 == 0) {  
        if (day > 29)  
            legal = FALSE;  
    }  
    else {  
        if (day > 28)  
            legal = FALSE;  
    }  
}
```

Right code

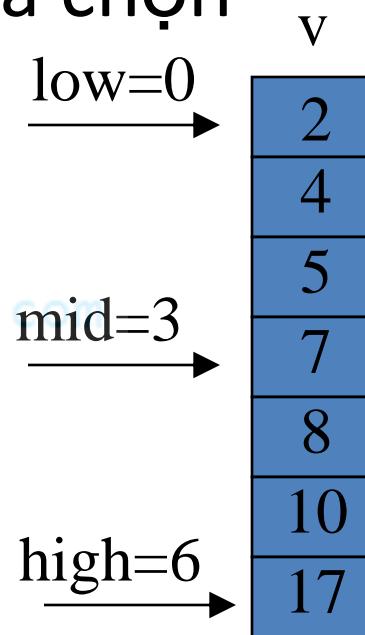
Structure: Indentation (cont.)

- Use “else-if” cho cấu trúc đa lựa chọn

- VD: Bước so sánh trong tìm kiếm nhị phân - binary search.

- Bad code

```
if (x < v[mid])
    cuu high = mid - 1;
else
    if (x > v[mid])
        low = mid + 1;
else
    return mid;
```



- Good code

```
if (x < v[mid])
    high = mid - 1;
else if (x > v[mid])
    low = mid + 1;
else
    return mid;
```

Structure: “Paragraphs”

thành các phần

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

int main(void)

/* Read a circle's radius from stdin, and compute and write its
   diameter and circumference to stdout. Return 0 if successful. */

{
    const double PI = 3.14159;
    int radius;
    int diam;
    double circum;

    printf("Enter the circle's radius:\n");
    if (scanf("%d", &radius) != 1)
    {
        fprintf(stderr, "Error: Not a number\n");
        exit(EXIT_FAILURE); /* or: return EXIT_FAILURE; */
    }
    ...
}
```

Structure: “Paragraphs”

- Dùng dòng trống để chia code thành các phần chính

```
diam = 2 * radius;  
circum = PI * (double)diam;  
  
printf("A circle with radius %d has diameter %d\n",  
      radius, diam);  
printf("and circumference %f.\n", circum);  
  
return 0;
```

```
}
```

Structure: Expressions

- Dùng các biểu thức dạng nguyên bản
 - VD: Kiểm tra nếu n thỏa mãn $j < n < k$
 - Bad code

if (! (n >= k) && ! (n <= j))

– Good code

if ((j < n) && (n < k))

- BT điều kiện có thể đọc như cách thức bạn viết thông thường
 - Đừng viết BT điều kiện theo kiểu mà bạn không bao giờ sử dụng

Structure: Expressions (cont.)

- Dùng () để tránh nhầm lẫn

- VD: Kiểm tra nếu n thỏa mãn $j < n < k$

- Moderately bad code

```
if (j < n && n < k)
```

- Moderately better code

```
if ((j < n) && (n < k))
```

- Nên nhóm các nhóm một cách rõ ràng

- Toán tử quan hệ (vd “ $>$ ”) có độ ưu tiên cao hơn các toán tử logic (vd “ $\&\&$ ”), **nhưng ai nhớ điều đó?**

Structure: Expressions (cont.)

- Dùng () để tránh nhầm lẫn (cont.)
 - VD: đọc và in các ký tự cho đến cuối tệp.
 - Wrong code (điều gì xảy ra ???)

```
while (c = getchar() != EOF)  
    putchar(c);
```

- Right code

```
while ((c = getchar()) != EOF)  
    putchar(c);
```

- Nên nhóm các nhóm một cách rõ ràng :

- *Toán tử Logic (“!=”) có độ ưu tiên cao hơn toán tử gán (“=”)*

Structure: Expressions (cont.)

- Đơn giản hóa các biểu thức phức tạp
 - VD: Xác định các ký tự tương ứng với các tháng của năm
 - Bad code

```
if ((c == 'J') || (c == 'F') || (c ==  
'M') || (c == 'A') || (c == 'S') || (c  
== 'O') || (c == 'N') || (c == 'D'))
```

- Good code

```
if ((c == 'J') || (c == 'F') ||  
(c == 'M') || (c == 'A') ||  
(c == 'S') || (c == 'O') ||  
(c == 'N') || (c == 'D'))
```

Nên xắp xếp các cơ cấu s

C Idioms

- Chú ý khi dùng ++, --

- VD: Set each array element to 1.0.

- Bad code (or perhaps just “so-so” code)

```
i = 0;  
while (i <= n-1) than cong . com  
    array[i++] = 1.0;
```

- Good

```
Code duong than cong . com  
for (i=0; i<n; i++)  
    array[i] = 1.0;
```

Naming

- Dùng tên gợi nhớ, có tính miêu tả cho các biến và hàm
 - VD : hovaten, **CONTROL**, **CAPACITY**
- Dùng tên nhất quán cho các biến cục bộ
 - VD, **i** (not **arrayIndex**) cho biến chạy vòng lặp
- Dùng chữ hoa, chữ thường nhất quán . com
 - VD., Buffer_Insert (Tên hàm)
CAPACITY (hằng số)
buf (biến cục bộ)
- Dùng phong cách nhất quán khi ghép từ
 - VD., **frontsize**, **FrontSize**, **front_size**
- Dùng động từ cho tên hàm
 - VD., docsolieu (), inkq(), **Check_Octal** (), ...

Comments

- Làm chủ ngôn ngữ
 - Hãy để chương trình tự diễn tả bản thân
 - Rồi...
- Viết chú thích để thêm thông tin
 - `i++; /* add one to i */`
- Chú thích các đoạn (“paragraphs”) code, đừng chú thích từng dòng
 - vd., “Sort array in ascending order”
- Chú thích dữ liệu tổng thể
 - Global variables, structure type definitions,
- Viết chú thích tương ứng với code!!!
 - Và thay đổi khi bản thân code changes. ☺

Comments (cont.)

- đoạn (“paragraphs”) không chú thích típ nσ dòng σ code

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

int main(void)
{
    /* Read a circle's radius from stdin, and compute and write its
       diameter and circumference to stdout. Return 0 if successful. */

    const double PI = 3.14159;
    int radius;
    int diam;
    double circum;

    /* Read the circle's radius. */
    printf("Enter the circle's radius:\n");
    if (scanf("%d", &radius) != 1)
    {
        fprintf(stderr, "Error: Not a number\n");
        exit(EXIT_FAILURE); /* or: return EXIT_FAILURE; */
    }
}
```

Comments (cont.)

```
/* Compute the diameter and circumference. */
diam = 2 * radius;
circum = PI * (double)diam;
        cuu duong than cong . com
/* Print the results. */
printf("A circle with radius %d has diameter
%d\n",
        radius, diam);
printf("and circumference %f.\n", circum);
        cuu duong than cong . com
return 0;
}
```

Function Comments

- Mô tả **những gì cần thiết để** gọi hàm 1 cách chính xác
 - Mô tả **Hàm làm gì**, chứ không phải **nó làm như thế nào**
 - Bản thân Code phải rõ ràng, dễ hiểu để biết cách nó làm việc...
 - Nếu không, hãy viết chú thích bên trong định nghĩa hàm
- Mô tả **đầu vào**: Tham số truyền vào, đọc file gì, biến tổng thể được dùng
- Mô tả **outputs**: giá trị trả về, tham số truyền ra, ghi ra files gì, các biến tổng thể mà nó tác động tới

Function Comments (cont.)

- Bad function comment

```
/* decomment.c */  
  
int main(void) {  
    /* Đọc 1 ký tự. Dựa trên ký tự ấy và trạng  
     * thái DFA hiện thời, gọi hàm xử lý trạng thái  
     * tương ứng. Lặp cho đến hết tệp end-of-file.  
 */  
  
    ...  
}  
...
```

- Describes **how the function works**

Function Comments (cont.)

- Good function comment

```
/* decomment.c */  
  
int main(void) {  
  
    /* Đọc 1 CT C qua stdin.  
       Ghi ra stdout với mỗi thing chú thích thaybằng 1 dấu  
       cách.  
       Trả về 0 nếu thành công, EXIT_FAILURE nếu không  
       thành công. */  
  
    ...  
}  
cuu duong than cong . com
```

- Describes what the function *does*

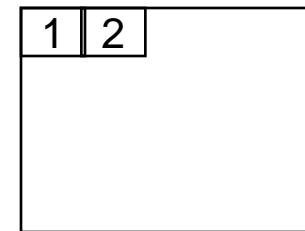
Modularity!!!

- Chương trình lớn viết khó hơn Ct nhỏ
- Trừu tượng hóa là chìa khóa để xử lý sự phức tạp
 - Abstraction cho phép LTV biết code làm gì, mà không cần biết làm như thế nào
- Ví dụ : hàm ở mức trừu tượng
 - Hàm sắp xếp 1 mảng các số nguyên
 - Character I/O functions : **getchar()** and **putchar()**
 - Mathematical functions : **sin (x)** and **sqrt (x)**

Bottom-Up Design is Bad

- 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



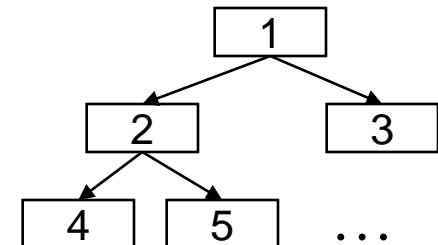
Top-Down Design is Good

- Top-down design 😊

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

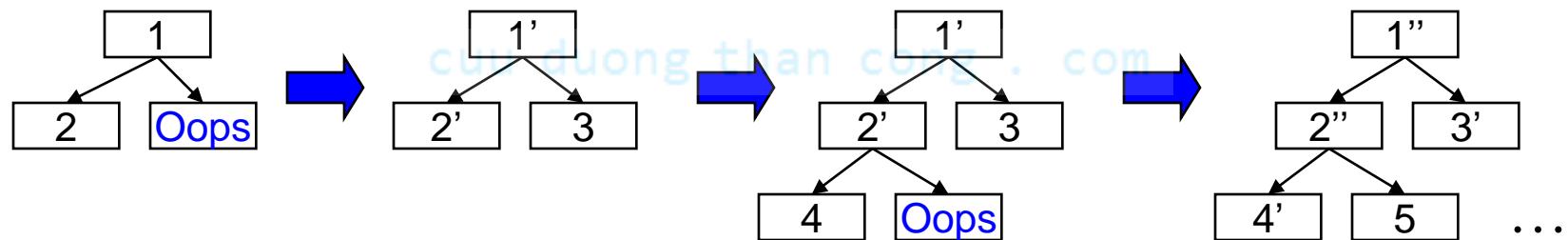
- Top-down design in programming

- Xây dựng sơ lược hàm main() bằng 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 được modul hóa 1 cách tự nhiên



Top-Down Design in Reality

- Thiết kế CT Top-down trong thực tiễn :
 - Định nghĩa hàm main() = pseudocode
 - Tinh chỉnh từng lệnh pseudocode
 - Nếu gặp sự cố Oops! Xem lại thiết kế, và...
 - Quay lại để tinh chỉnh pseudocode đã có, và tiếp tục
 - Lặp lại (mostly) ở mức sâu hơn, cụ thể hơn, cho đến khi các hàm được định nghĩa xong



Ví dụ: Text Formatting

- Mục tiêu :
 - Minh họa good program và programming style
 - Đặc biệt là modul hóa mức hàm và top-down design
 - Minh họa cách đi từ vấn đề đến viết code
 - Ôn lại và mô tả cách xây dựng CT C
- Text formatting
 - Đầu vào: ASCII text, với hàng loạt dấu cách và phân dòng
 - Đầu ra: Cùng nội dung, nhưng ~~căn trái và căn phải~~ ~~com~~
 - Dồn các từ tối đa có thể trên 1 dòng 50 ký tự
 - Thêm các dấu cách cần thiết giữa các từ để căn phải
 - Không cần căn phải dòng cuối cùng
 - Để đơn giản hóa, giả định rằng :
 - 1 từ kết thúc bằng dấu cách space, tab, newline, hoặc end-of-file
 - Không có từ nào quá 20 ký tự

Ví dụ về Input and Output

I
N
P
U
T

Tune every heart and every voice.
Bid every bank withdrawal.
Let's all with our accounts rejoice.
In funding Old Nassau.
In funding Old Nassau we spend more money every year.
Our banks shall give, while we shall live.
We're funding Old Nassau.

O
U
T
P
U
T

Tune every heart and every voice. Bid every bank withdrawal. Let's all with our accounts rejoice. In funding Old Nassau. In funding Old Nassau we spend more money every year. Our banks shall give, while we shall live. We're funding Old Nassau.

Thinking About the Problem

- Khái niệm “từ”
 - Chuỗi các ký tự không có khoảng trắng, tab xuống dòng, hoặc EOF
 - Tất cả các ký tự trong 1 từ phải được in trên cùng 1 dòng
- Làm sao để đọc và in dc các từ
 - Đọc các ký tự từ stdin cho đến khi gặp space, tab, newline, or EOF
 - In các ký tự ra stdout tiếp theo bởi các dấu space(s) or newline
- Nếu đầu vào lộn xộn thì thế nào ?
 - Cần loại bỏ các dấu spaces thừa, các dấu tabs, và newlines từ input
- Làm sao có thể căn phai ?
 - Ta không biết được số dấu spaces cần thiết cho đến khi đọc hết các từ
 - Cần phải lưu lại các từ cho đến khi có thể in được trọn vẹn 1 dòng
- Nhưng, Bao nhiêu space cần phải thêm vào giữa các từ?
 - Cần ít nhất 1 dấu space giữa các từ riêng biệt trên 1 dòng
 - Có thể thêm 1 vài dấu spaces để phủ kín 1 dòng

Writing the Program

- Key constructs
 - Từ - Word
 - Dòng - Line
- Các bước tiếp theo
 - Viết pseudocode cho hàm main()
 - Tinh chỉnh
- Lưu ý :
Chú thích hàm và một số dòng trống được bỏ qua vì những hạn chế không gian

Trình tự thiết kế là lý tưởng

Trong thực tế, nhiều backtracking sẽ xảy ra

The Top Level

- pseudocode hàm main()...

```
int main(void) {  
    <Xóa dòng>  
    for (;;) {  
        <Đọc 1 từ>  
        if (<Hết từ>) {  
            <In dòng không cần căn phải>  
            return 0;  
        }  
        if (<Từ không vừa dòng hiện tại>) {  
            <In dòng có căn lề phải>  
            <Xóa dòng>  
        }  
        <Thêm từ vào dòng>  
    }  
    return 0;  
}
```

Reading a Word

```
#include <stdio.h>
enum {MAX_WORD_LEN = 20};
int main(void) {
    char word[MAX_WORD_LEN + 1];
    int wordLen;
    < Xóa dòng >
    for (;;) {
        wordLen = ReadWord(word);
        if (< Hết từ >) {
            < In dòng không cần căn phải >
            return 0;
        }
        if (< Từ không vừa dòng hiện tại >) {
            < In dòng có căn lề phải >
            < Xóa dòng >
        }
        < Thêm từ vào dòng >
    }
    return 0;
}
```

- <Đọc 1 từ> nghĩa là gì? Việc này khá phức tạp nên cần tách thành 1 hàm riêng ...

```
int ReadWord(char *word) {
    < Bỏ qua whitespace >
    < Lưu các ký tự cho đến MAX_WORD_LEN của từ >
    < Trả về độ dài từ >
}
```

Reading a Word (cont.)

```
int ReadWord(char *word) {
    int ch, pos = 0;

    /* Bỏ qua whitespace. */
    ch = getchar();
    while ((ch == ' ') || (ch == '\n') || (ch == '\t'))
        ch = getchar();

    /* Lưu các ký tự vào từ cho đến MAX_WORD_LEN.*/
    while ((ch != ' ') && (ch != '\n') && (ch != '\t') && (ch != EOF)) {
        if (pos < MAX_WORD_LEN) {
            word[pos] = (char)ch;
            pos++;
        }
        ch = getchar();
    }
    word[pos] = '\0';

    /* Trả về độ dài từ.*/
    return pos;
}
```

Reading a Word (cont.)

```
int ReadWord(char *word) {  
    int ch, pos = 0;  
  
    /* Bỏ qua whitespace. */  
    ch = getchar();  
    while (IsWhitespace(ch))  
        ch = getchar();  
  
    /* Lưu các ký tự vào từ cho đến MAX_WORD_LEN. */  
    while (!IsWhitespace(ch) && (ch != EOF)) {  
        if (pos < MAX_WORD_LEN) {  
            word[pos] = (char)ch;  
            pos++;  
        }  
        ch = getchar();  
    }  
    word[pos] = '\0';  
  
    /* trả về độ dài từ. */  
    return pos;  
}
```

- Hmm. ReadWord() chứa 1 vài đoạn code lặp lại => tách thành 1 hàm riêng :
IsWhitespace(ch)

```
int IsWhitespace(int ch) {  
    return (ch == ' ') || (ch == '\n') || (ch == '\t');  
}
```

Copying ~ Word

```
#include <stdio.h>
#include <string.h>
enum {MAX_WORD_LEN = 20};
enum {MAX_LINE_LEN = 50};
int main(void) {
    char word[MAX_WORD_LEN + 1];
    int wordLen;
    char line[MAX_LINE_LEN + 1];
    int lineLen = 0;
    <Xóa dòng> cuu duong than cong . com
    for (;;) {
        wordLen = ReadWord(word);
        if (<Hết từ>) {
            <In dòng không căn lề>
            return 0;
        }
        if (<Từ không vừa dòng> ) {
            < In dòng có căn lề >
            <Xóa dòng>
        }
        AddWord(word, line, &lineLen);
    }
    return 0;
}
```

Quay lại main().
<Thêm từ vào dòng>

có nghĩa là gì ?

- Tạo 1 hàm riêng cho việc đó :

**AddWord(word, line,
&lineLen)**

void AddWord(const char *word, char
*line, int *lineLen) {
 <Nếu dòng đã chứa 1 số từ, thêm 1 dấu
trắng>
 strcat(line, word);
 (*lineLen) += strlen(word);
}

Saving a Word (cont.)

- AddWord()

```
void AddWord(const char *word, char *line, int *lineLen) {  
  
    /* Nếu dòng đã chứa 1 số từ, thêm 1 dấu trắng. */  
    if (*lineLen > 0){  
        line[*lineLen] = ' ';  
        line[*lineLen + 1] = '\0';  
        (*lineLen)++;  
    }  
  
    strcat(line, word);  
    (*lineLen) += strlen(word);  
}
```

Printing the Last Line

```
...
int main(void) {
    char word[MAX_WORD_LEN + 1];
    int wordLen;
    char line[MAX_LINE_LEN + 1];
    int lineLen = 0;
    <Xóa dòng>
    for (;;) {
        wordLen = ReadWord(word);
        /* Nếu hết từ, in dòng không căn lề. */
        if ((wordLen == 0) && (lineLen > 0)) {
            puts(line);
            return 0;
        }
        if (<Từ không vừa dòng>) {
            <In dòng có căn lề>
            <Xóa dòng>
        }
        AddWord(word, line, &lineLen);
    }
    return 0;
}
```

- <Hết từ> và <In dòng không căn lề> nghĩa là gì ?
- Tạo các hàm để thực hiện

Deciding When to Print

```
...
int main(void) {
    char word[MAX_WORD_LEN + 1];
    int wordLen;
    char line[MAX_LINE_LEN + 1];
    int lineLen = 0;
    <Xóa dòng>
    for (;;) {
        wordLen = ReadWord(word);
        /* If no more words, print line
         * with no justification. */
        if ((wordLen == 0) && (lineLen > 0)) {
            puts(line);
            return 0;
        }
        /* Nếu từ không vừa dòng, thì ... */
        if ((wordLen + 1 + lineLen) > MAX_LINE_LEN) {
            <In dòng có căn lề>
            < Xóa dòng >
        }
        AddWord(word, line, &lineLen);
    }
    return 0;
}
```

- <Từ không vừa dòng>
Nghĩa là gì?

Printing with Justification

- Bây giờ , đến trong tâm của CT. <In dòng có căn lề> nghĩa là gì ?
- Rõ ràng hàm này cần biết trong dòng hiện tại có bao nhiêu từ. Vì vậy ta thêm **numWords** vào hàm main ...

```
...
int main(void) {
    ...
    int numWords = 0;
    <Xóa dòng>
    for (;;) {
        ...
        /* Nếu từ không vừa dòng, thì... */
        if ((wordLen + 1 + lineLen) > MAX_LINE_LEN) {
            WriteLine(line, lineLen, numWords);
            <Xóa dòng>
        }
        AddWord(word, line, &lineLen);
        numWords++;
    }
    return 0;
}
```

Printing with Justification (cont.)

- Viết pseudocode cho WriteLine()...

```
void WriteLine(const char *line, int lineLen, int numWords) {  
  
    <Tính số khoảng trắng dư thừa cho dòng>  
    cuu duong than cong . com  
    for (i = 0; i < lineLen; i++) {  
        if (<line[i] is not a space>)  
            <Print the character>  
        else {  
            <Tính số khoảng trắng cần bù thêm>  
            cuu duong than cong . com  
            <In 1 space, cộng thêm các spaces cần bù>  
            cuu duong than cong . com  
            <Giảm thêm không gian và đếm số từ>  
        }  
    }  
}
```

Printing with Justification (cont.)

```
void WriteLine(const char *line, int lineLen, int numWords)
{
```

```
    int extraSpaces, spacesToInsert, i, j;
```

```
    /* Tính số khoảng trắng dư thừa cho dòng. */
```

```
    extraSpaces = MAX_LINE_LEN - lineLen;
```

```
    for (i = 0; i < lineLen; i++) {
```

```
        if (line[i] != ' ')
```

```
            putchar(line[i]);
```

```
        else {
```

```
            /* Tính số khoảng trắng cần thêm. */
```

```
            spacesToInsert = extraSpaces / (numWords - 1);
```

```
            /* In 1 space, cộng thêm các spaces phụ. */
```

```
            for (j = 1; j <= spacesToInsert + 1; j++)
```

```
                putchar(' ');
```

```
            /* Giảm bớt spaces và đếm từ. */
```

```
            extraSpaces -= spacesToInsert;
```

```
            numWords--;
```

```
}
```

```
}
```

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

CuuDuongThanCong.com

- Hoàn tất hàm WriteLine()...

Số lượng các khoảng trắng

VD:

Nếu extraSpaces = 10 và numWords = 5, thì space bù sẽ là 2, 2, 3, and 3 tương ứng

Clearing the Line

- Cuối cùng. <Xóa dòng> nghĩa là gì ?
- Tuy đơn giản, nhưng ta cũng xd thành 1 hàm

```
...
int main(void) {
    ...
    int numWords = 0;
    ClearLine(line, &lineLen, &numWords);
    for (;;) {  
        ...
        /* If word doesn't fit on this line, then... */
        if ((wordLen + 1 + lineLen) > MAX_LINE_LEN) {
            WriteLine(line, lineLen, numWords);
            ClearLine(line, &lineLen, &numWords);
        }

        addWord(word, line, &lineLen);
        numWords++;
    }
    return 0;
}
```

```
void ClearLine(char *line, int  
*lineLen, int *numWords) {  
    line[0] = '\0';  
    *lineLen = 0;  
    *numWords = 0;  
}
```

Modularity: Tóm tắt ví dụ

- **Với người sử dụng CT**

- n xộn
 - Output: Cùng nội dung, nhưng trình bày căn lề trái, phải, rõ ràng, sáng sửa

- **Giữa các phần của CT**

- Các hàm xử lý từ : Word-handling functions
 - Các hàm xử lý dòng : Line-handling functions
 - main() function

- **Lợi ích của modularity**

- Đọc code: dễ ràng, qua các mẫu nhỏ, riêng biệt
 - Testing : Test từng hàm riêng biệt
 - Tăng tốc độ: Chỉ tập trung vào các O(hanf chậm)
 - Mở rộng: Chỉ thay đổi các phần liên quan

The “justify” Program

```
#include <stdio.h>
#include <string.h>
enum {MAX_WORD_LEN = 20};
enum {MAX_LINE_LEN = 50};
int IsWhitespace(int ch) {
    return (ch == ' ') || (ch == '\n') || (ch == '\t');
}
int ReadWord(char *word) {
    int ch, pos = 0;
    ch = getchar(); cuu duong than cong . com
    while (IsWhitespace(ch))
        ch = getchar();
    while (!IsWhitespace(ch) && (ch != EOF)) {
        if (pos < MAX_WORD_LEN) {
            word[pos] = (char)ch;
            pos++;
        }
        ch = getchar(); cuu duong than cong . com
    }
    word[pos] = '\0';
    return pos;
}
```

```
void ClearLine(char *line, int *lineLen, int
*numWords) {
    line[0] = '\0';
    *lineLen = 0;
    *numWords = 0;
}
void AddWord(const char *word, char *line, int
*lineLen) {  
    cuu duong than cong . com
    if (*lineLen > 0) {
        line[*lineLen] = ' ';
        line[*lineLen + 1] = '\0';
        (*lineLen)++;
    }  
    cuu duong than cong . com
    strcat(line, word);
    (*lineLen) += strlen(word);
}
```

```
void WriteLine(const char *line, int lineLen, int numWords) {  
    int extraSpaces, spacesToInsert, i, j;  
    extraSpaces = MAX_LINE_LEN - lineLen;  
    for (i = 0; i < lineLen; i++) {  
        if (line[i] != ' ')  
            putchar(line[i]);  
        else {  
            spacesToInsert = extraSpaces / (numWords - 1);  
            for (j = 1; j <= spacesToInsert + 1; j++)  
                putchar(' ');  
            extraSpaces -= spacesToInsert;  
            numWords--;  
        }  
    }  
    putchar('\n');  
}
```

```
int main(void) {
    char word[MAX_WORD_LEN + 1];
    int wordLen;
    char line[MAX_LINE_LEN + 1];
    int lineLen = 0;
    int numWords = 0;
    ClearLine(line, &lineLen, &numWords);
    for (;;) {
        wordLen = ReadWord(word);
        if ((wordLen == 0) && (lineLen > 0)) {
            puts(line);
            break;
        }
        if ((wordLen + 1 + lineLen) > MAX_LINE_LEN) {
            WriteLine(line, lineLen, numWords);
            ClearLine(line, &lineLen, &numWords);
        }
        AddWord(word, line, &lineLen);
        numWords++;
    }
    return 0;
}
```

Optimizing C and C++ Code

- **Đặt kích thước mảng = bội của 2**

Với mảng, khi tạo chỉ số, trình dịch thực hiện các phép nhân, vì vậy, hãy đặt kích thước mảng bằng bội số của 2 để phép nhân có thể dc chuyển thành phép toán dịch chuyển nhanh chóng

- **Đặt các case labels trong phạm vi hẹp**

Nếu số case label trong câu lệnh switch nằm trong phạm vi hẹp, trình dịch sẽ biến đổi thành if – else - if lồng nhau, mà tạo thành 1 bảng các chỉ số, như vậy thao tác sẽ nhanh hơn

- **Đặt các trường hợp thường gấp trong lệnh switch lên đầu**

Khi số các trường hợp tuyển chọn là nhiều và tách biệt, trình dịch sẽ biến lệnh switch thành các nhóm if – else –if lồng nhau. Nếu bố trí các case thường gấp lên trên, việc thực hiện sẽ nhanh hơn

- **Tái tạo các switch lớn thành các switches lồng nhau**

Khi số cases nhiều, hãy chủ động chia chúng thành các switch lồng nhau, nhóm 1 gồm những cases thường gấp, và nhóm 2 gồm những cases ít gấp=> kết quả là các phép thử sẽ ít hơn, tốc độ nhanh hơn

Optimizing C and C++ Code (tt)

- **Minimize local variables**

Các biến cục bộ được cấp phát và khởi tạo khi hàm đc gọi, và giải phóng khi hàm kết thúc, vì vậy mất thời gian

- **Khai báo các biến cục bộ trong phạm vi nhỏ nhất**

- **Hạn chế số tham số của hàm**

- **Với các tham số và giá trị trả về > 4 bytes, hãy dùng tham chiếu**

- **Đừng định nghĩa giá trị trả về, nếu không sử dụng (void)**

- **Lưu ý ví trí của tham chiếu tới code và data**

Các bộ xử lý dữ liệu hoặc mã giữ được tham chiếu trong bộ nhớ cache để tham khảo về sau, nếu được nó từ bộ nhớ cache. Những tài liệu tham khảo bộ nhớ cache được nhanh hơn. Do đó nó nên mã và dữ liệu đang được sử dụng cùng nhau thực sự nên được đặt với nhau . Điều này với object trong C + + là đương nhiên. Với C ? (Không dùng biến tổng thể, dùng biến cục bộ ...)

Optimizing C and C++ Code (tt)

- Nên dùng int thay vì char hay short (mất thời gian convert), nếu biết int không âm, hãy dùng unsigned int
- Hãy định nghĩa các hàm khởi tạo đơn giản
- Thay vì gán, hãy khởi tạo giá trị cho biến
- Hãy dùng danh sách khởi tạo trong hàm khởi tạo

```
Employee::Employee(String name, String designation) {      m_name =  
    name;
```

```
          cuu.duong.than.cong . com  
    m_designation = designation;  
}
```

```
/* === Optimized Version === */
```

```
Employee::Employee(String name, String designation): m_name(name),  
    m_designation (designation) { }
```

- Đừng định nghĩa các hàm ảo tùy hứng : "just in case" virtual functions
- Các hàm gồm 1 đến 3 dòng lệnh nên định nghĩa inline

Một vài ví dụ tối ưu mã C, C++

```
typedef unsigned int uint;  
uint div32u (uint a) {  
    return a / 32;  
}  
int div32s (int a){  
    return a / 32; // return a >>5;  
}
```

```
switch ( queue ) {  
    case 0 : letter = 'W'; break;  
    case 1 : letter = 'S'; break;  
    case 2 : letter = 'U'; break;  
}
```

Hoặc có thể là :

```
if ( queue == 0 )  
    letter = 'W';
```

```
else if ( queue == 1 )  
    letter = 'S';
```

```
else letter = 'U';
```

```
static char *classes="WSU";  
letter = classes[queue];
```

$(x \geq \min \& \& x < \max)$ có thể chuyển thành

$(\text{unsigned})(x - \min) < (\max - \min)$

```
int fact1_func (int n) {
```

```
    int i, fact = 1;
```

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

```
    return (fact);      cuu duong than cong . com
```

```
}
```

```
int fact2_func(int n) {
```

```
    int i, fact = 1;
```

```
    for (i = n; i != 0; i--) fact *= i;
```

```
    return (fact);      cuu duong than cong . com
```

```
}
```

Fact2_func nhanh hơn, vì phép thử \neq đơn giản hơn \leq

Tối ưu đoạn code sau :

```
found = FALSE;  
for(i=0;i<10000;i++) {  
    if( list[i] == -99 ) {  
        found = TRUE;           ← i while...  
    }  
}  
if( found ) printf("Yes, there is a -99. !\n");
```

Floating_point

So sánh :

$x = x / 3.0;$

Và

$x = x * (1.0/3.0) ;$

? [cuu duong than cong . com](http://cuuduongthancong.com)

(biểu thức hằng được thực hiện ngay khi dịch)

Hãy dùng float thay vì double

Tránh dùng sin, exp và log (chậm gấp 10 lần *)

Lưu ý : nếu x là float hay double thì : $3 * (x / 3) \neq x$.

Thậm chí thứ tự tính toán cũng quan trọng: $(a + b) + c \neq a + (b + c)$.

- Tránh dùng ++, -- trong biểu thức lặp , vd
while (n--) {...}
- Dùng $x * 0.5$ thay vì $x / 2.0$.
cuu duong than cong . com
- $x+x+x$ thay vì $x*3$
- Mảng 1 chiều nhanh hơn mảng nhiều chiều
- Tránh dùng đệ quy
cuu duong than cong . com

Chương 4

Một số cấu trúc dữ liệu và giải thuật căn bản

cuu-duong-than-cong . com

Phần 4.1 Đệ qui (4LT – 2BT)

cuu-duong-than-cong . com

SE-SolCT

KTLT 4-1.1

1. Đệ qui

- 1.1 Khái niệm về đệ qui
- 1.2 Các loại đệ qui
- 1.3 Mô tả đệ qui các cấu trúc dữ liệu
- 1.4 Mô tả đệ qui các giải thuật
- 1.5 Các dạng đệ qui đơn giản thường gặp

Khái niệm Đ/n đệ qui

- Một mô tả/định nghĩa về một đối tượng gọi là đệ qui nếu trong mô tả/định nghĩa đó ta lại sử dụng chính đối tượng này.
- Tức là mô tả đối tượng qua chính nó.
 - Mô tả đệ qui tập số tự nhiên N :
 - Số 1 là số tự nhiên (1 -N)
 - Số tự nhiên bằng số tự nhiên cộng 1.
 - Mô tả đệ qui cấu trúc ds(list) kiểu T :
 - Cấu trúc rỗng là một ds kiểu T.
 - Ghép nối một thành phần kiểu T(nút kiểu T) với một ds kiểu T ta có một ds kiểu T.
 - Mô tả đệ qui cây gia phả: Gia phả của một người bao gồm người đó và gia phả của cha và gia phả của mẹ

Ví dụ

■ Định nghĩa không đệ qui n!:

● $n! = n * (n-1) * \dots * 1$

■ Định nghĩa đệ qui:

● $n! = \begin{cases} 1 & \text{nếu } n=0 \\ n * (n-1)! & \text{nếu } n>0 \end{cases}$

■ Mã C++:

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

■ Mô tả đệ qui thủ tục sắp tăng dãy

● $a[m:n]$ (dãy $a[m], a[m+1], \dots, a[n]$) bằng phương pháp Sort_Merge (SM):
 $SM(a[m:n]) \equiv Merge (SM(a[m : (n+m) \text{ div } 2]), SM(a[(n+m) \text{ div } 2 + 1 : n]))$

➤ Với : $SM(a[x : x])$ là thao tác rỗng (không làm gì cả).

➤ $Merge (a[x : y], a[(y+1) : z])$ là thủ tục trộn 2 dãy tăng $a[x : y], a[(y+1) : z]$ để
được một dãy $a[x : z]$ tăng.

■ Mô tả đệ qui gồm hai phần

- Phần neo: trường hợp suy biến (cá biệt) của đối tượng

Ví dụ: 1 là số tự nhiên, cấu trúc rỗng là danh sách kiểu T , $0 != 1, \dots$

- Phần qui nạp: mô tả đối tượng (giải thuật) thông qua chính đối tượng (giải thuật) đó một cách trực tiếp hoặc gián tiếp.

Ví dụ:

$$n! = n * (n - 1) !$$

$$SM(a[m:n]) \equiv Merge(SM(a[m:(m+n) \text{ div } 2]), SM(a[(m+n) \text{ div } 2 + 1 : n]))$$

■ Đệ qui gồm hai loại:

- Đệ qui trực tiếp
- Đệ qui gián tiếp

Giải thuật đệ qui

- Nếu ta có 1 lời giải S cho bài toán P, ta lại sử dụng lời giải ấy cho bài toán P' giống P nhưng kích cỡ nhỏ hơn thì lời giải S đó gọi là lời giải đệ qui.
- Biểu diễn giải thuật đệ qui

$$P \Leftrightarrow P[S , P]$$

Điều kiện dừng

- Biểu diễn tổng quát

$$P \Leftrightarrow \text{if } B \text{ } P[S , P]$$

$$\text{hoặc } P \quad P[S , \text{if } B \text{ } P]$$

- Chương trình con đệ qui: Khi ta cài đặt giải thuật đệ qui, ta có chương trình đệ qui (tự nó gọi lại chính nó: $P \Rightarrow P'$)
 - Hàm đệ qui
 - Thủ tục đệ qui

Mô tả đệ qui các giải thuật

- Dãy số Fibonaci(FIBO) : $\{ FIBO(n) \} \equiv 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, \dots$
 - $FIBO(0) = FIBO(1) = 1$;
 - $FIBO(n) = FIBO(n-1) + FIBO(n-2)$; với $n >= 2$
- Giải thuật đệ qui tính $FIBO(n)$ là:

```
FIBO(n)
if ((n = 0) or (n = 1)) return 1 ;
else return (FIBO(n-1) + FIBO(n-2));
```

Các dạng đệ quy đơn giản thường gặp

■ Đệ quy tuyến tính: là dạng đệ quy trực tiếp đơn giản nhất có dạng

```
P() {  
    If (B) thực hiện S;  
    else { thực hiện S* ; gọi P }  
}
```

- Với S , S* là các thao tác không đệ quy . . . com
- Ví dụ:Hàm FAC(n) tính số hạng n của dãy n!

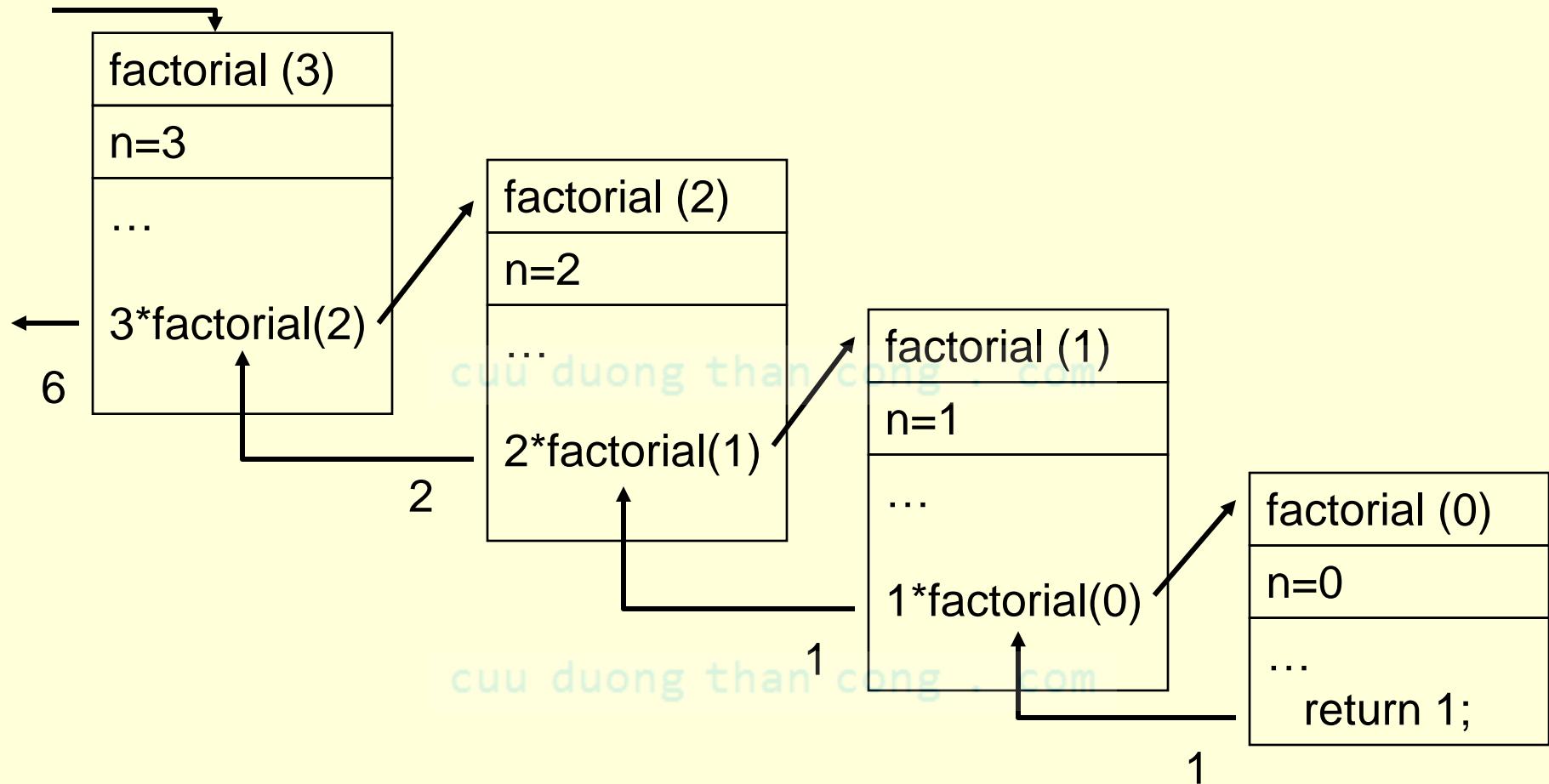
Dạng hàm trong ngôn ngữ mã giả:

```
{ Nếu n = 0 thì FAC = 1 ; /* trường hợp neo */  
  Ngược lại FAC = n * FAC(n-1) }
```

Dạng hàm trong C++ :

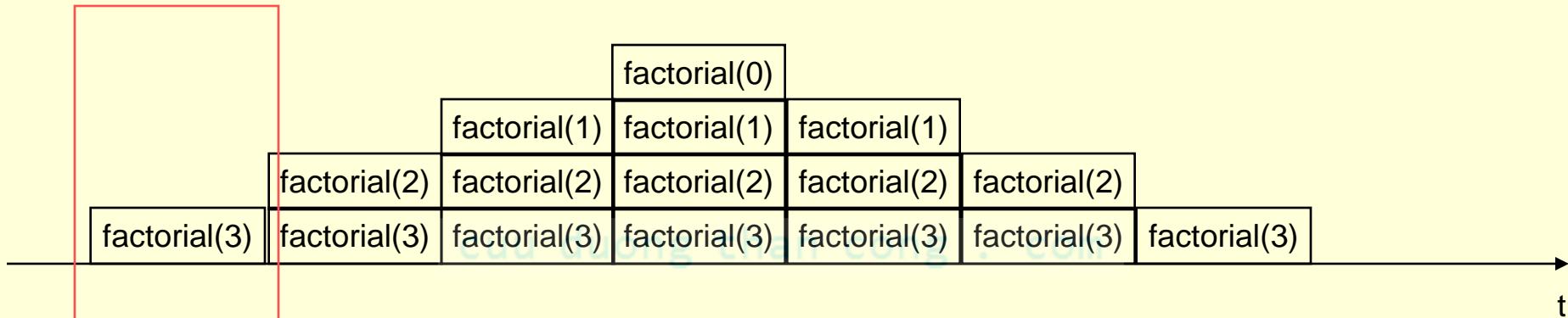
```
int FAC( int n )  
{  
    if ( n == 0 ) return 1 ;  
    else return ( n * FAC(n-1) ) ;  
}
```

Thi hành hàm tính giai thừa

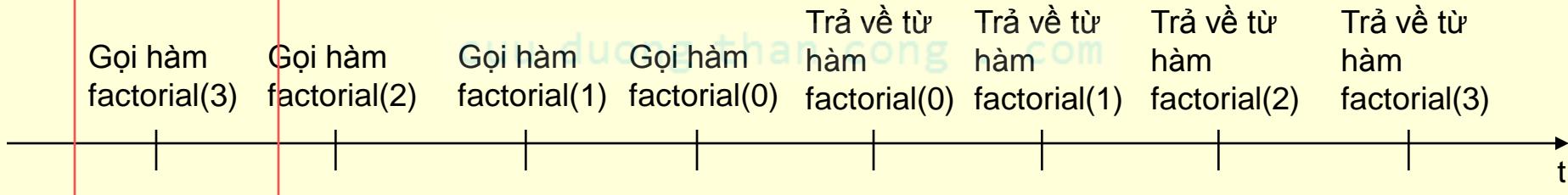


Trạng thái hệ thống khi thi hành hàm tính giai thừa

Stack hệ thống



Thời gian hệ thống



Các dạng đệ qui đơn giản thường gặp (tiếp)

■ Đệ qui nhị phân: là đệ qui trực tiếp có dạng như sau

$P() \{$

If (B) thực hiện S;

else { thực hiện S ; gọi P ; gọi P...}*

$\}$

cuu duong than cong . com

● Với S , S* là các thao tác không đệ qui .

● Ví dụ: Hàm FIBO(n) tính số hạng n của dãy FIBONACCI

Dạng hàm trong C++ :

int F(int n)

cuu duong than cong . com

{ if (n < 2) return 1 ;

else return (F(n -1) + F(n -2)) ; }

Các dạng đệ qui đơn giản thường gặp (tiếp)

- Đệ qui phi tuyến: là đệ qui trực tiếp mà lời gọi đệ qui được thực hiện bên trong vòng lặp.

```
P () {  
    for (<giá trị đầu> to <giá trị cuối>)  
    {      thực hiện S ;  
          if ( thỏa điều kiện dừng ) then thực hiện S*;  
          else gọi P; }  
}
```

- Với S , S* là các thao tác không đệ qui .

Ví dụ: Cho dãy { An } xác định theo công thức truy hồi :

$$A_0 = 1 ; A_n = n^2 A_0 + (n-1)^2 A_1 + \dots + 2^2 A_{n-2} + 1^2 A_{n-1}$$

Dạng hàm đệ qui tính An trên ngôn ngữ C++ là:

```
int A( int n ) {  
    if ( n == 0 ) return 1 ;  
    else {  
        int tg = 0 ;  
        for (int i = 0 ; i < n ; i++ ) tg = tg + sqr(n-i) *A(i);  
        return ( tg ) ;  
    }  
}
```

3 bước để tìm giải thuật đệ qui

■ Tham số hóa bài toán .

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

■ Tìm các trường hợp neo (“Trivial”) cùng giải thuật giải tương ứng

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

Vd : $FAC(1) = 1$

$USCLN(a,b) = b$ nếu a chia hết cho b

■ Tìm giải thuật giải trong trường hợp tổng quát bằng phân rã bài toán theo kiểu đệ qui.

3 bước (tiếp)

■ Phân rã bài toán tổng quát theo phương thức đệ qui

● Phân rã bài toán thành các bài toán giống BT

cuuduongthancong . com

Top-Down!

● Ví dụ

$$FAC(n) = n * FAC(n - 1)$$

$$Tmax(a[1:n]) = \max(Tmax(a[1:(n-1)]), a[n])$$

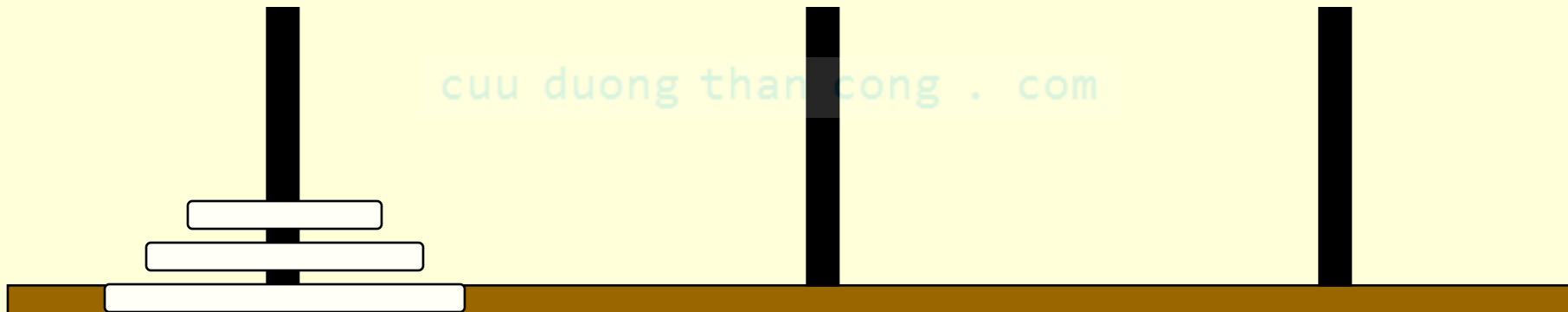
Một số bài toán giải bằng đệ quy

- Bài toán tháp Hà Nội
- Bài toán chia phần thưởng
- Bài toán hoán vị

Bài toán Tháp Hà nội

■ Luật:

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



Với chồng gồm n đĩa cần $2^n - 1$ lần chuyển.

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

$$- T = (2^{64} - 1) * t = 1.84 \cdot 10^{19} t$$

$$- Với t = 1/100 s thì T = 5.8 \cdot 10^9 \text{ năm} = 5.8 \text{ tỷ năm} .$$

Bài toán Tháp Hà nội

■ Hàm đệ quy:

- Chuyển (n-1) đĩa trên đỉnh của cột start sang cột temp
- Chuyển 1 đĩa (cuối cùng) của cột start sang cột finish
- Chuyển n-1 đĩa từ cột temp sang cột finish



Bài toán Tháp Hà nội

■ Giải bài toán bằng đệ quy

● Thông số hóa bài toán

- Xét bài toán ở mức tổng quát nhất : chuyển n ($n \geq 0$) đĩa từ cột A sang cột B lấy cột C làm trung gian .
- $\text{THN}(n, A, B, C) \rightarrow$ với 64 đĩa gọi $\text{THN}(64, A, B, C)$
- n sẽ là thông số quyết định bài toán – n là tham số điều khiển

● Trường hợp suy biến và cách giải

- Với $n = 1$: $\text{THN}(1, A, B, C)$
 - ◆ Giải thuật giải bt $\text{THN}(1, A, B, C)$ là thực hiện chỉ 1 thao tác cơ bản : Chuyển 1 đĩa từ A sang B (ký hiệu là $\text{Move}(A, B)$).
- $\text{THN}(1, A, B, C) \equiv \{ \text{Move}(A, B) \}$
- $\text{THN}(0, A, B, C) \equiv \{ \varphi \}$

Bài toán Tháp Hà nội

Phân rã bài toán

- Ta có thể phân rã bài toán TH N (k,A,B,C) : chuyển k đĩa từ cột A sang cột B lấy cột C làm trung gian thành dãy tuần tự 3 công việc sau :

- Chuyển (k -1) đĩa từ cột A sang cột C lấy cột B làm trung gian :
 - THN (k -1,A,C,B) (bài toán THN với n = k-1, A = A , B = C , C = B)
- Chuyển 1 đĩa từ cột A sang cột B : Move (A, B) (thao tác cơ bản).
- Chuyển (k -1) đĩa từ cột C sang cột B lấy cột A làm trung gian :
 - THN(k -1,C,B,A) (bài toán THN với n = k-1 , A = B , B = A , C = C) .

Vậy giải thuật trong trường hợp tổng quát ($n > 1$) là:
THN(n ,A,B,C)

{

THN ($n -1$,A,C,B) ;
Move (A, B) ;
THN ($n -1$,C,B,A) ;

}

Bài toán Tháp Hà nội – Mã C++

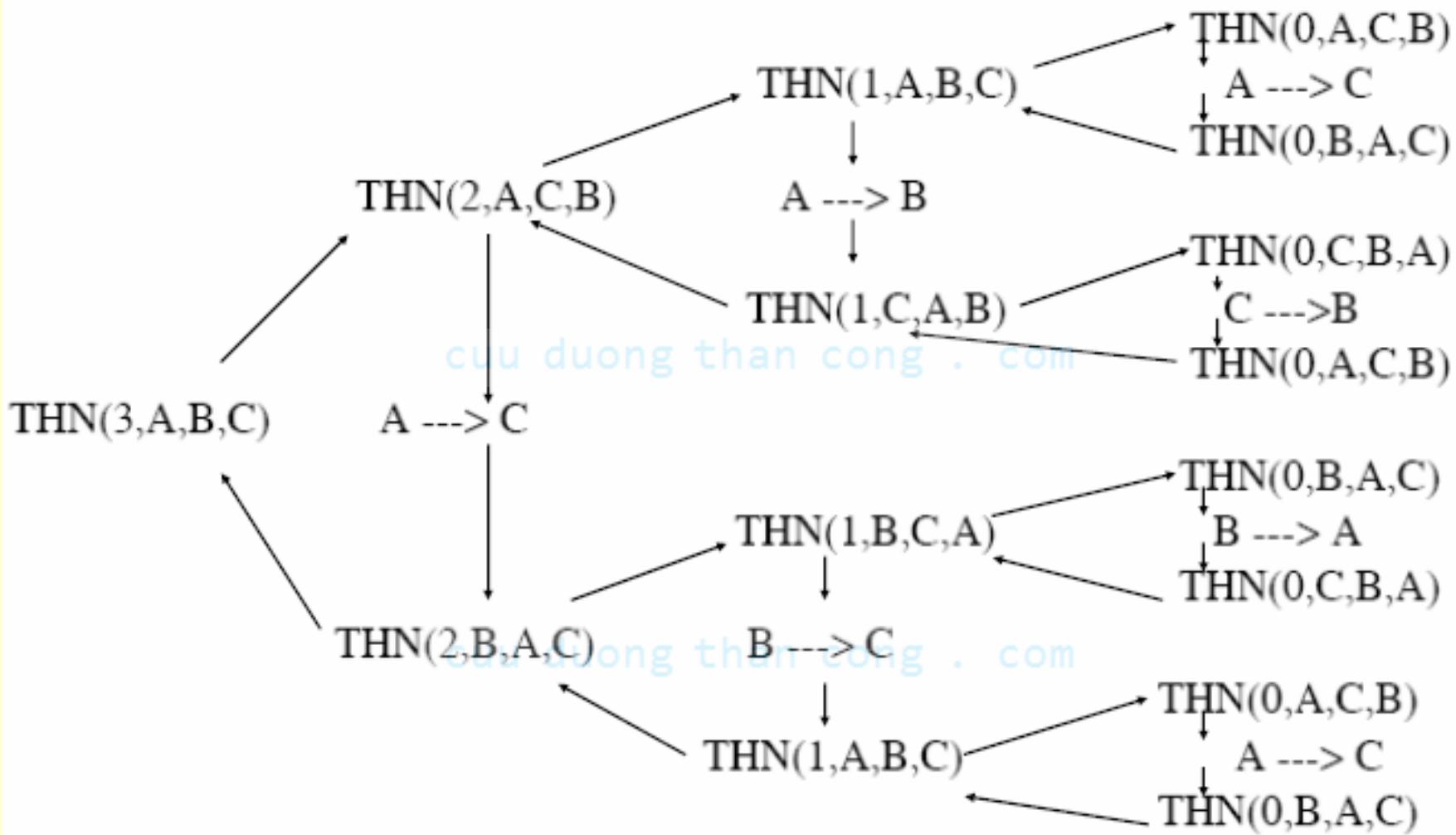
```
void move(int count, int start, int finish, int temp) {  
    if (count > 0) {  
        move(count - 1, start, temp, finish);  
        cout << "Move disk " << count << " from " << start  
            << " to " << finish << "." << endl;  
        move(count - 1, temp, finish, start);  
    }  
}
```

Lời gọi c/o

Lối gọi c/1

Lời gợi c/2

Lời gợi ý c/3



Bài toán chia phần thưởng

- Có **100 phần thưởng** đem chia cho **12 học sinh giỏi đã được xếp hạng**. Có bao nhiêu cách khác nhau để thực hiện cách chia?
- Tìm giải thuật giải bài toán bằng phương pháp đệ quy.

cuu duong than cong . com

Bài toán chia phần thưởng (tự đọc)

■ Giải bài toán bằng đệ quy

- Nhìn góc độ bài toán tổng quát: Tìm số cách chia m vật (phần thưởng) cho n đối tượng (học sinh) có thứ tự.

➤ PART(m ,n)

➤ N đối tượng đã được sắp xếp $1,2,\dots,n$

➤ Số là số phần thưởng mà i nhận được

$$S_i \geq 0$$

$$S_1 \geq S_2 \geq \dots \geq S_n$$

$$S_1 + S_2 + \dots + S_n = m$$

➤ Ví dụ: [cuu duong than cong . com](http://cuuduongthancong.com)

Với $m = 5$, $n = 3$ ta có 5 cách chia sau :

5 0 0 , 4 1 0 , 3 2 0 , 3 1 1 , 2 2 1

Tức là $\text{PART}(5,3) = 5$

Các trường hợp suy biến

- $m = 0$: mọi học sinh đều nhận được 0 phần thưởng .
 $PART(0 , n) = 1$ với mọi n
- $n = 0 , m <> 0$: không có cách chia
 $PART(m , 0) = 0$ với mọi $m <> 0$.

Phân rã bài toán trong trường hợp tổng quát

- $m < n$: ~~n -m~~ học sinh xếp cuối sẽ luôn ~~không~~ không nhận được gì cả trong mọi cách chia .
Vậy: $n > m$ thì $PART(m , n) = PART(m , m)$
- $m \geq n$: là tổng
 - Học sinh cuối cùng không có phần thưởng
 $PART(m , n -1)$
 - Học sinh cuối cùng có ít nhất 1
 $PART(m -n , n)$Vậy: $m > n \Rightarrow PART(m , n) = PART(m , n -1) + PART(m -n , n)$

Dạng hàm PART trong NN LT C++

```
int PART( int m , int n ) {  
    if ((m == 0 ) || (n == 0) ) return 1 ;  
    else if(m < n) retrun ( PART(m , m )) ;  
    else  
        return ( PART(m , n -1 ) + PART( m -n , n ) ) ;  
}
```

cuu duong than cong . com

Bài toán tìm tất cả hoán vị của một dãy các phần tử (giảng lướt)

■ Thông số hóa bài toán.

- Gọi HV(v, m)

- v : array[1 .. N] of T

- m :integer ; m <= N

- T là một kiểu dữ liệu

- Ví dụ: N = 4 , A[1] = 1 , A[2] = 2 , A[3] = 3 , A[4] = 4 thì lời gọi HV(A ,3) xuất tất cả hoán vị của A có được bằng cách hoán vị 3 phần tử đầu (có 6 h vị) :

1 2 3 4	1 3 2 4	3 2 1 4
2 1 3 4	3 1 2 4	2 3 1 4

■ Trường hợp neo

i m = 1 : HV(v,1): xuất v

HV(v,1) ≡Display(v) //for (k= 1 to N do) Display(v[k])

Phân rã bài toán

- Giữ nguyên các phần tử cuối $V[m], \dots, V[N]$ hoán vị $m-1$ phần tử đầu
 - gọi đệ quy $HV(V, m-1)$
- Đổi chỗ $V[m]$ cho $V[m-1]$, giữ nguyên các phần tử cuối $V[m], \dots, V[N]$ hoán vị $m-1$ phần tử đầu
 - gọi đệ quy $HV(V, m-1)$
- Đổi chỗ $V[m]$ cho $V[m-2]$, giữ nguyên các phần tử cuối $V[m], \dots, V[N]$ hoán vị $m-1$ phần tử đầu
 - gọi đệ quy $HV(V, m-1)$
-
-
- Đổi chỗ $V[m]$ cho $V[2]$, giữ nguyên các phần tử cuối $V[m], \dots, V[N]$ hoán vị $m-1$ phần tử đầu
 - gọi đệ quy $HV(V, m-1)$.
- - Đổi chỗ $V[m]$ cho $V[1]$, giữ nguyên các phần tử cuối $V[m], \dots, V[N]$ hoán vị $m-1$ phần tử đầu
 - gọi đệ quy $HV(V, m-1)$.

Bài toán tìm tất cả hoán vị của một dãy các phần tử

```
HV(V,m) ≡{  
    SWAP( V[m],V[m] ) ; HV(V,m -1) ;  
    SWAP( V[m],v[m-1] ) ; HV(V,m -1) ;  
    SWAP( V[m],v[m-2] ) ; HV(V,m -1) ;  
    .....  
    .....  
    SWAP (V[m],v[2] ) ; HV(V,m -1) ;  
    SWAP( V[m],v[1] ) ; HV(V,m -1) ;  
}  
SWAP(x , y ) là thủ tục hoán đổi giá trị của 2 đối tượng dữ liệu x ,y )  
Vậy :HV(V , m ) ≡ for (k = m;k>0; k--) {  
    SWAP( V[m], V[k] ) ;  
    HV(V,m -1) ;  
} ;
```

```

const size = Val ; // Val là hằng giá trị
typedef typebase vector[size] ; // typebase là một kiểu dữ liệu có thứ tự

void Swap( typebase &x , typebase &y) {
    typebase t ;
    t = x ; x = y ; y = t ;
}
void print( const vector &A) {
    for(int j= 0 ; j <size ; j++ ) cout<< A[j] ;
    cout << "....";
}
void HV( const vector &V , int m) {
    if (m == 1 ) print( V );
    else for(int k = m-1 ; k >= 0 ; k--) {
        swap(V[m-1] ,V[k]);
        HV(V,m-1) ;
    }
}

```

Lời giải Đệ qui khác của bài toán Hoán vị (giảng cho SV)

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#define STOP 27 // phím esc
const MAX =100;
int a[MAX];
char b[MAX];// mảng danh dấu
int shv, n;

Hoanvi (int k)
{ int i,j;
for (i=1;i<=n;i++)
if (b[i])
{ a[k] = i;
b[i] = 0;
if (k==n)
{ // Đã chọn đủ n số => 1 hoán vị
shv++;
printf("\n Hoán vị %2d là : ", shv);
for (j=1; j <=n;j++)
printf("%d", a[j]);
}
else // goi tang len 1 so
Hoanvi (k+1);
b[i] = 1;
}
return 0;
}
```

Lời giả không đê qui của bài toán Hoán vị (tiếp)

```
■ main ()  
■ { char ch;  
■ int i;  
■ do  
■ { // danh dau mang b  
■     for (i=1;i <= MAX;i++)    b[i] = 1;  
■     clrscr();  cuu duong than cong . com  
■     printf("\n n=");  
■     scanf("%d", &n);  
■     shv = 0;  
■     if ( n > 0)  
■         Hoanvi(1);  
■         printf("\n Chu tuon thanh: ESC>"); com  
■         ch = getch();  
■     } while (ch !=STOP);  
■ }
```

KHỬ ĐỆ QUI

- 1 Cơ chế thực hiện đê qui
- 2 Tổng quan về khử đê qui
- 3 Các trường hợp khử đê qui đơn giản

Cơ chế thực hiện đệ qui

- Trạng thái của tiến trình xử lý một giải thuật: nội dung các biến và lệnh cần thực hiện kế tiếp.
- Với tiến trình ~~sử dụng thanh công~~ ^{sử dụng thanh công} xử lý một giải thuật đệ qui ở từng thời điểm thực hiện, cần lưu trữ cả các trạng thái xử lý đang còn dang dở .

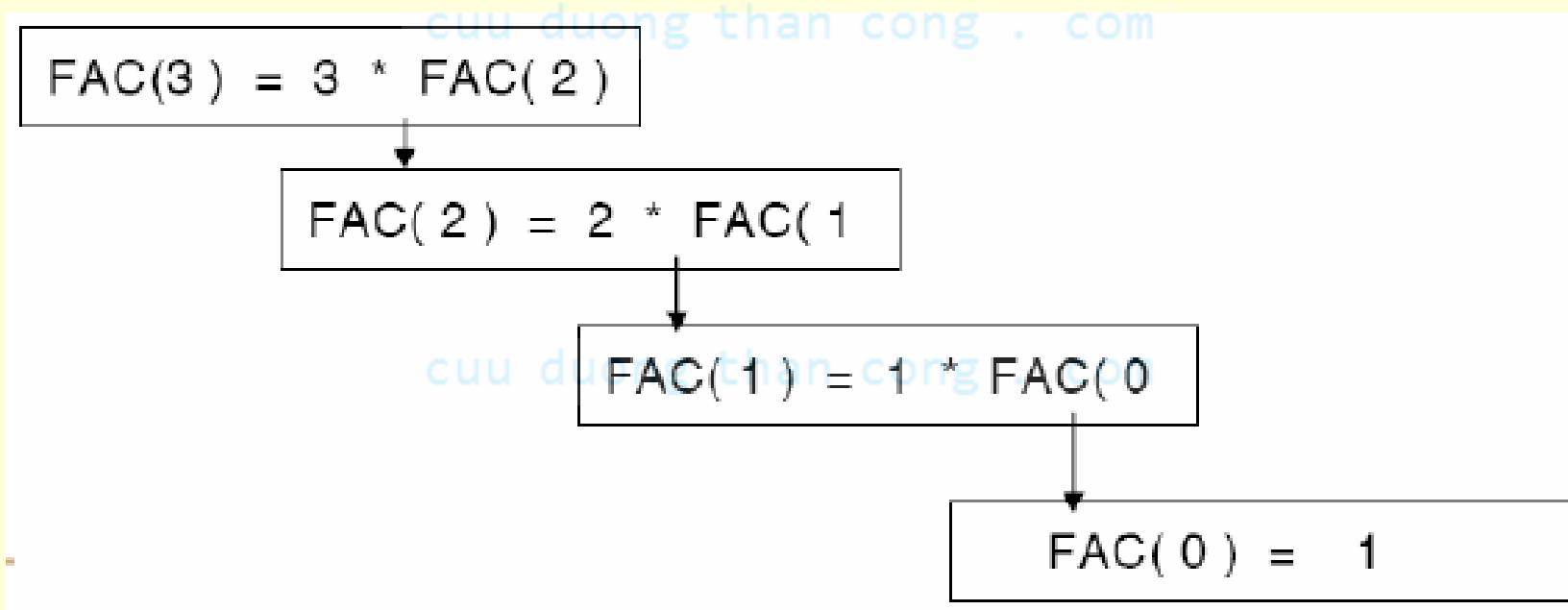
[cuu duong than cong . com](http://cuuduongthancong.com)

Xét giải thuật giai thừa

-Giải thuật

```
FAC ( n ) ≡ if( n = 0 ) retrun 1;  
else retrun ( n * FAC (n-1));
```

-Sơ đồ thực hiện



Xét thủ tục đệ qui tháp Hà Nội THN (n , X , Y , Z)

-

trung gian

– Giải thuật :

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

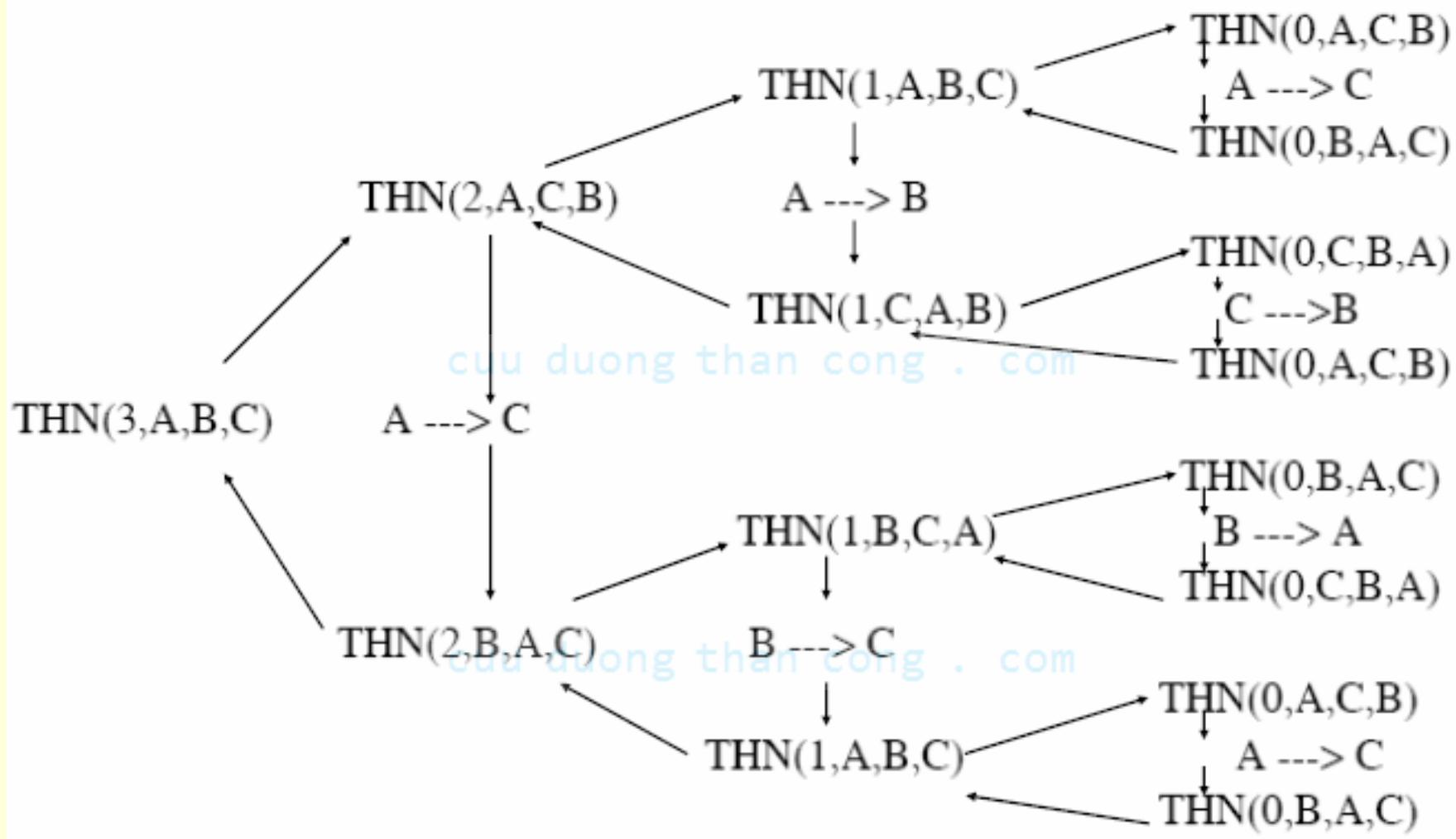
– Sơ đồ thực hiện THN(3,A,B,C)

Lời gợi ý/0

Lời gợi ý/1

Lời gợi ý/2

Lời gợi ý/3



Nhận xét

- Lời gọi đệ qui sinh ra lời gọi đệ qui mới cho đến khi gặp trường hợp suy biến (neo)
 - Ở mỗi lần gọi phải lưu trữ thông tin trạng thái con dang dở của tiến trình xử lý ở thời điểm gọi. Số trạng thái này bằng số lần gọi chưa được hoàn tất .
cuuduongthancong.com
 - Khi thực hiện xong (hoàn tất) một lần gọi, cần khôi phục lại toàn bộ thông tin trạng thái trước khi gọi .
 - Lệnh gọi cuối cùng (ứng với trường hợp neo) sẽ được hoàn tất đầu tiên.
-
- ng: cấu trúc Stack (LIFO)

Tạo ngăn xếp S

- Thủ tục Creatstack(S) : Tạo S rỗng .
- Thủ tục Push(x,S) : thêm x vào đỉnh stack S
 - (x là dữ liệu kiểu đơn giản hoặc có cấu trúc)
- Thủ tục Pop(x,S) : Lấy giá trị đang lưu ở đỉnh S
 - Lưu trữ vào x
 - Loại bỏ giá trị này khỏi S (lùi đỉnh S xuống một mức)
- Hàm Empty(S): (kiểu boolean) Kiểm tra tính rỗng của S : cho giá trị đúng nếu S rỗng , sai nếu không.

Cai dat stack :

```
#define MAX 100
typedef struct {
    int top;
    int nodes(MAX);
} stack;      cuu duong than cong . com
int Empty(stack *ps) {
    if (ps->top == -1)
        return (true);
    return(false);
}
```

```
void Push(stack *ps , int x) {  
    if (ps->top ==MAX -1) {  
        printf("\n stack full");  
        return;  
    }  
    ps->top +=1 ;  
    ps->nodes[ps->top] =x;  
}  
  
int Pop(stack *ps) {  
    if(Empty(ps) {  
        printf("\n stack is empty");  
        return (0);  
    }  
    return(ps->nodes[ps->top--]);
```

Tổng quan về khử đệ qui

t code

- Nhược điểm: tốn không gian nhớ và thời gian xử lý.
- Mọi giải thuật đệ qui đều có thể thay thế bằng một giải thuật không đệ qui.
- Sơ đồ xây dựng chương trình cho một bài toán khó khi ta không tìm được giải thuật không đệ qui thường là:
 - i) Dùng quan niệm đệ qui để tìm giải thuật cho bài toán .
 - ii) Mã hóa giải thuật đệ qui .
 - iii) Khử đệ qui để có được một chương trình không đệ qui .
- Tuy nhiên : khử đệ qui không phải bao giờ cũng dễ => trong nhiều trường hợp ta cũng phải chấp nhận sử dụng chương trình đệ qui.

1.Khử đệ qui bằng vòng lặp

A. *a trị của dãy dữ liệu mô tả bằng hồi qui*

•Ý tưởng

–Xét một vòng lặp trong đó sử dụng 1 tập hợp biến $W = (V, U)$

i) Tập hợp U các biến bị thay đổi

ii) V là các biến còn lại.

–Dạng tổng quát của vòng lặp là: $W = W_0 ; \{ W_0 = (U_0, V_0) \}$

while $C(U)$ do $U := g(W)$

–Gọi U_0 là trạng thái của U ngay trước vòng lặp

– U_k với $k > 0$ là trạng thái của U sau lần lặp thứ k (giả sử còn lặp đến lần k)

U_0 mang các giá trị được gán ban đầu

$U_k = g(W) = g(U_{k-1}, V_0) = f(U_{k-1})$ với $k = 1 .. n$

Với n là lần lặp cuối cùng, tức $C(U_k)$ đúng với mọi $k < n$, $C(U_n)$ sai

–Sau vòng lặp W mang nội dung (U_n, V_0) .

Giải thuật hồi qui thường gấp

$$f(n) = C \text{ khi } n = n_0 \quad (C \text{ là một hằng})$$
$$= g(n, f(n-1)) \text{ khi } n > n_0$$

• Ví dụ:

- Hàm giải thừa $FAC(n) = n! = 1$ khi $n = 0$
 $= n * FAC(n-1)$ khi $n > 0$

– Tổng n số đầu tiên của dãy đan dấu sau :

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

$$S(k) = \begin{cases} 1 & \text{khi } k = 1 \\ S(k-1) + (-1)^{k+1} * (2k-1) & \text{với } k > 1 \end{cases}$$

.Giải thuật đệ quy tính giá trị $f(n)$

```
f(n) = if(n = no) return C ;  
else return (g(n,f(n -1)) ;
```

•Giải thuật lặp tính giá trị $f(n)$

```
k := no ; F := C ;  
{ F = f(no) }  
while( k < n ) {  
    k := k +1 ;  
    F := g(k,F) ;  
}  
return F;
```

•Trong trường hợp này :

```
W = U = ( k ,F )  
Wo = Uo = ( no,C )  
C(U) = ( k < n)  
f(W) = f(U) = f(k,F) = (k+1,g(k,F)))
```

Khử đệ qui với hàm tính giai thừa

```
long int FAC ( int n ) {  
    int k = 0 ;  
    long int F = 1 ;  
    while ( k < n ) F = ++k * F ;  
    return (F) ;  
}
```

Với hàm tính S(n)

```
int S ( 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 ) ;  
}
```

2.Các thủ tục đệ qui dạng đệ qui đuôi

- Xét thủ tục P dạng :

$P(X) \equiv \text{if } B(X) \text{ then } D(X)$

else {

A(X) ;

P(f(X)) ;

}

- Trong đó: X là tập biến (một hoặc một bộ nhiều biến).
- P(X) là thủ tục đệ qui phụ thuộc X
- A(X) ; D(X) là các thao tác không đệ qui
- f(X) là hàm biến đổi X

- Xét quá trình thi hành P(X) :

- gọi Po là lần gọi P thứ 0 (đầu tiên) P(X)
- P1 là lần gọi P thứ1 (lần 2) P(f(X))
- Pi là lần gọi P thứ i (lần i + 1) P(f(f(...f(X)...))
- (P(fi(X)) hợp i lần hàm f)

- Gọi Pi nếu B(fi(X))

- (false) { A và gọi Pi+1 }
- (true) { D }

- Giả sử P được gọi đúng n +1 lần . Khi đó ở trong lần gọi cuối cùng (thứ n) Pn thì B(fn(X)) =true , lệnh D được thi hành và chấm dứt thao tác gọi thủ tục P .

Sơ đồ thực hiện giải thuật trên bằng vòng lặp:

while (! B(X)) {
 A(X) ;
 X = f(X) ;
}
D(X) ;

Ví dụ:

Để đổi 1 số nguyên không âm Y ở cơ số 10 sang dạng cơ số k ($2 \leq k \leq 9$)

- Dùng mảng A [n]
- Convert(x,m) để tạo dãy giá trị: A[0] , A[1] , . . . , A[m]
- Giải thuật

Convert(n,m) ≡ if n != 0 {

A[m] = n % k ;

Convert(n/k , m -1) ;

}

– Trong ví dụ này ta có

- X là(n, m) ;
- B(X) là biểu thức boolean not(n <> 0)
- A(X) là lệnh gán A[m] := n%k ;
- f(X) là hàm $f(n,m) = (n/k , m -1)$;
- D(X) là lệnh rỗng

– Đoan lệnh lặp tương ứng với thủ tục Convert(x,m) là:

while (n != 0) {

A[m] = n % k ; //A(X)

n = n / k ; // X := f(X)

m = m -1 ;

Last update 8-2010

}

CuuDuongThanCong.com

SE-SolICT

<https://fb.com/tailieudientucntt>

KTLT 4-1.48

Ví dụ: Tìm ước số chung lớn nhất của hai số

- Giải thuật đệ qui

USCLN(m , n , var us) \equiv if (n = 0) then us := m
else USCLN(n , m mod n , us) ;

=> t trên c

unsigned USCLN2(int a, int b)

```
{ // Dùng đệ qui theo định nghĩa của USCLN
    if ((a % b)== 0) return b;
    else return USCLN2(b, a % b);
}
```

qui

```
unsigned USCLN1(int a, int b)
{// Dùng vòng lặp theo thuật toán O'clid
while (a !=b)
{ if (a > b) a-=b;
else b-=a;
}
return b;
}
```

ng Stack

- Để thực hiện một chương trình con đệ qui thì hệ thống phải tổ chức vùng lưu trữ thỏa qui tắc LIFO (Stack).=> So sánh với gọi CTC thông thường.
- Vậy ta chủ động tạo ra cấu trúc dữ liệu stack đặc dụng cho từng chương trình con đệ qui cụ thể phù hợp cơ chế LIFO.

A. Đệ qui chỉ có một lệnh gọi trực tiếp

•Đệ qui có dạng sau:

$P(X) \equiv \text{if } C(X) \text{ D}(X)$

else {

 A(X) ;

 P(f(X)) ;

 B(X) ;

}

X là một biến đơn hoặc biến véc tơ.

C(X) là một biểu thức boolean của X.

A(X) , B(X) , D(X):không đệ qui

f(X) là hàm của X (hàm đơn điệu giảm)

Giải thuật thực hiện P(X) với việc sử dụng Stack có dạng :

P(X) ≡ {

 Create_Stack (S) ; (tạo stack S)

 while(not(C(X))

 { A(X) ; cuu duong than cong . com

 Push(S,X) ;

 X := f(X) ;

 }

 D(X) ;

 while (not Empty(S)) {

 POP(S,X) ;

 B(X) ;

 }

}

- Ví dụ: Thủ tục đệ qui chuyển biểu diễn số từ cơ số thập phân sang nhị phân có dạng :

```
Binary(m) ≡ if ( m > 0 ) {  
    Binary( m div 2 ) ;  
    write( m mod 2 ) ;  
}
```

- Trong trường hợp này :

X là m .

P(X) là Binary(m) .

A(X) ; D(X) là lệnh rỗng .

B(X) là lệnh Write(m mod 2) ;

C(X) là (m <= 0) .

f(X) = f(m) = m div 2

Giải thuật thực hiện Binary(m) không đệ qui là:

Binary (m)

{ Create_Stack (S) ;

 while (m > 0) {

 sdu := m mod 2 ;

 Push(S,sdu);
 m := m div 2 ;

 }

 while (not Empty(S)) {

 POP(S,sdu) ;

 Display(sdu);
 }

}

}

B. Thủ tục đệ quy với hai lần gọi đệ quy

-Đệ quy có dạng sau

```
P(X) ≡ if C(X)  D(X) ;  
else {  
    A(X) ; P(f(X)) ;  
    B(X) ; P(g(X)) ;  
}
```

-Thuật toán khử đệ quy tương ứng với thủ tục đệ quy

P(X) là:{

Creat_Stack (S) :

Push (S, (X,1)) ;

do

{ **while (not C(X)) {**

 A(X) ;

Push (S, (X,2)) ;

 X := f(X) ;

 }

D(X) ;

POP (S, (X,k)) ;

if (k <> 1) {

 B(X) ;

 X := g(X) ;

 }

} while (k = 1) ;

Khử đệ qui thủ tục Tháp Hà Nội .

- Dạng đệ qui

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

• Giải thuật không đệ qui tương đương là:

THN (n, X, Y, Z)

{ Creat_Stack (S) ;

 Push (S ,(n,X,Y,Z,1)) ;

 do

 while (n > 0) {

 Push (S ,(n,X,Y,Z,2)) ;

 n := n -1 ;

 Swap (Y,Z) ;

 }

 POP (S,(n,X,Y,Z,k)) ;

 if (k <> 1) {

 Move (X ,Z) ;

 n := n -1 ;

 Swap (X,Y) ;

 }

 } while (k = 1) ;

}

Bài tập

- Liệt kê mọi tập con củaa tập $1,2,3,\dots,n$, với n nhập từ bàn phím
- Liệt kê mọi hoán vị của Từ COMPUTER (mở rộng, 1 từ bất kỳ nhập từ bàn phím)
- Một nhà thám hiểm đem theo 1 cái túi với trọng lượng tối đa là B. Có n đồ vật cần mang theo, mỗi đồ vật có trọng lượng a_i và giá trị c_i tương ứng.Hãy viết CT tìm cách bỏ vào túi các đồ vật sao cho giá trị sử dụng là lớn nhất.
- Bài toán Người du lịch : 1 người du lịch muốn đi thăm các thành phố khác nhau. Xuất phát tại 1 thành phố nào đó, họ muốn lần lượt qua tất cả các thành phố (1 lân) rồi trở lại thành phố ban đầu.Biết chi phí đi lại từ thành phố I đến J là C_{ij} . Hãy tìm hành trình với tổng chi phí thấp nhất
8 x 8
sao cho chúng không ăn được nhau.

Bài toán 8 con Hậu – Giải thuật

Algorithm Solve

Input trạng thái bàn cờ

Output

1. **if** trạng thái bàn cờ chứa đủ 8 con hậu
 - 1.1. In trạng thái này ra màn hình
2. **else**
 - 2.1. **for** mỗi ô trên bàn cờ mà còn an toàn
 - 2.1.1. thêm một con hậu vào ô này
 - 2.1.2. dùng lại giải thuật Solve với trạng thái mới
 - 2.1.3. bỏ con hậu ra khỏi ô này

End Solve



Vét cạn

Bài toán 8 con Hậu – Thiết kế phương thức

```
bool Queens :: unguarded(int col) const;
```

Post: Returns **true** or **false** according as the square in the first unoccupied row (row count) and column col is not guarded by any queen.

```
void Queens :: insert(int col);
```

Pre: The square in the first unoccupied row (row count) and column col is not guarded by any queen.

Post: A queen has been inserted into the square at row count and column col; count has been incremented by 1.

```
void Queens :: remove(int col);
```

Pre: There is a queen in the square in row count – 1 and column col.

Post: The above queen has been removed; count has been decremented by 1.

```
bool Queens :: is_solved() const;
```

Post: The function returns **true** if the number of queens already placed equals board_size; otherwise, it returns **false**.

Bài toán 8 con Hậu – Thiết kế dữ liệu đơn giản

```
const int max_board = 30;

class Queens {
public:
    Queens(int size);
    bool is_solved() const;
    void print( ) const;
    bool unguarded(int col) const;
    void insert(int col);
    void remove(int col);
    int board_size; // dimension of board = maximum number of queens
private:
    int count; // current number of queens = first unoccupied row
    bool queen_square[max_board][max_board];
};
```

Bài toán 8 con Hậu – Mã C++

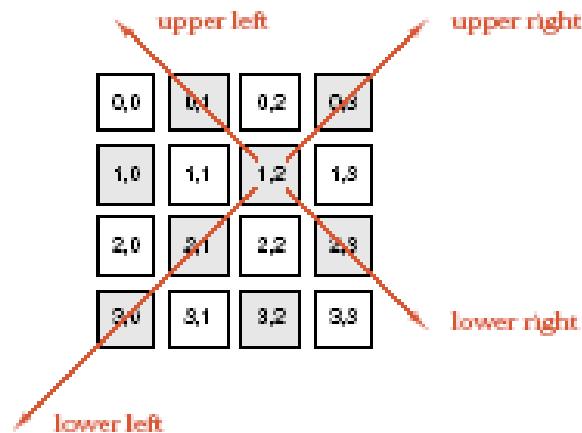
```
void Queens :: insert(int col) {  
    queen_square[count++][col] = true;  
}
```

```
bool Queens :: unguarded(int col) const {  
    int i;  
    bool ok = true;  
    for (i = 0; ok && i < count; i++) //kiểm tra tại một cột  
        ok = !queen_square[i][col];  
    //kiểm tra trên đường chéo lên  
    for (i = 1; ok && count - i >= 0 && col - i >= 0; i++)  
        ok = !queen_square[count - i][col - i];  
    //kiểm tra trên đường chéo xuống  
    for (i = 1; ok && count - i >= 0 && col + i < board_size; i++)  
        ok = !queen_square[count - i][col + i];  
    return ok;  
}
```

Bài toán 8 con Hậu – Góc nhìn khác

0 →	0,0	0,1	0,2	0,3
1 →	1,0	1,1	1,2	1,3
2 →	2,0	2,1	2,2	2,3
3 →	3,0	3,1	3,2	3,3

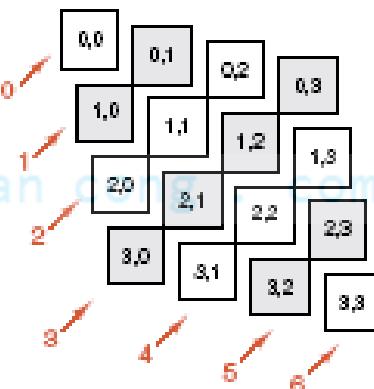
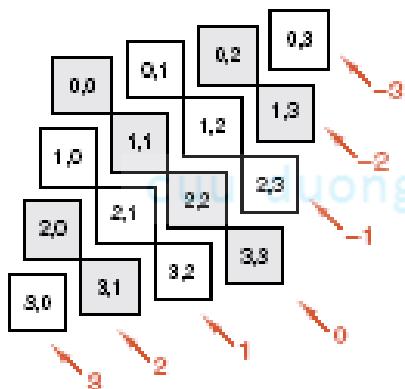
0,0	0,1	0,2	0,3
1,0	1,1	1,2	1,3
2,0	2,1	2,2	2,3
3,0	3,1	3,2	3,3



(a) Rows

(b) Columns

(c) Diagonal directions



difference = row - column

sum = row + column

Last update 8-2010

SE-SOIU1

KTLT 4-1.64

Bài toán 8 con Hậu – Thiết kế mới

```
const int max_board = 30;
class Queens {
public:
    Queens(int size);
    bool is_solved() const;
    void print() const;
    bool unguarded(int col) const;
    void insert(int col);
    void remove(int col);
    int board_size;
private:
    int count;
    bool col_free[max_board];
    bool upward_free[2 * max_board - 1];
    bool downward_free[2 * max_board - 1];
    int queen_in_row[max_board]; //column number of queen in each row
};
```

Bài toán 8 con Hậu – Mã C++ mới

```
Queens :: Queens(int size) {  
    board_size = size;  
    count = 0;  
    for (int i = 0; i < board_size; i++)  
        col_free[i] = true;  
    for (int j = 0; j < (2 * board_size - 1); j++)  
        upward_free[j] = true;  
    for (int k = 0; k < (2 * board_size - 1); k++)  
        downward_free[k] = true;  
}
```

```
void Queens :: insert(int col) {  
    queen_in_row[count] = col;  
    col_free[col] = false;  
    upward_free[count + col] = false;  
    downward_free[count - col + board_size - 1] = false;  
    count++;  
}
```

Chương 4

Một số cấu trúc dữ liệu và giải thuật căn bản

cuu duong than cong . com

qui

u (5LT-3BT)

cuu duong than cong . com

1. Các khái niệm cơ bản

Cấu trúc dữ liệu

- Cấu trúc dữ liệu là cách tổ chức và thao tác có hệ thống trên dữ liệu
- Một cấu trúc dữ liệu :
 - Mô tả
 - Các dữ liệu cấu thành
 - Mối liên kết về mặt cấu trúc giữa các dữ liệu đó
 - Xác định các thao tác trên dữ liệu đó

1. Các khái niệm cơ bản

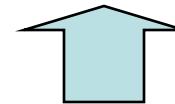
Kiểu dữ liệu

- **Kiểu dữ liệu cơ bản (primitive data type)**
 - Đại diện cho các dữ liệu giống nhau, không thể phân chia nhỏ hơn được nữa
 - Thường được các ngôn ngữ lập trình định nghĩa sẵn
 - Ví dụ:
 - C/C++: int, long, char, boolean, v.v.
 - Thao tác trên các số nguyên: + - * / ...
- **Kiểu dữ liệu có cấu trúc (structured data type)**
 - Được xây dựng từ các kiểu dữ liệu (cơ bản, có cấu trúc) khác
 - Có thể được các ngôn ngữ lập trình định nghĩa sẵn hoặc do lập trình viên tự định nghĩa

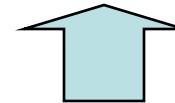
1. Các khái niệm cơ bản

Dữ liệu, kiểu dữ liệu, cấu trúc dữ liệu

Machine Level Data Storage 0100110001101001010001



Primitive Data Types 28 3.1415 'A'



Basic Data Structures array



High-Level Data Structures stack queue list

hash table tree

SE-SolCT

KLT4-2.4

05/04/2019

Last Update 8-2010
CuuDuongThanCong.com

<https://fb.com/tailieudientucntt>

II. Cấu trúc dữ liệu

cuu duong than cong . com

- c)
- **Danh sách**
- **Cây**
- **Bảng băm** cuu duong than cong . com

1. Danh sách (list)

- Danh sách :
 - Tập hợp các phần tử cùng kiểu
 - Số lượng các phần tử của danh sách không cố định
 - Phân loại:
 - Danh sách tuyến tính:
 - Có phần tử đầu tiên, phần tử cuối cùng
 - Thứ tự trước / sau của các phần tử được xác định rõ ràng
 - Danh sách không tuyến tính: các phần tử trong danh sách không được sắp thứ tự
 - Có nhiều hình thức lưu trữ danh sách
 - Sử dụng vùng các ô nhớ liên tiếp trong bộ nhớ → danh sách kế tiếp
 - Sử dụng vùng các ô nhớ không liên tiếp trong bộ nhớ → danh sách mốc nối
 - Danh sách nối đơn
 - Danh sách nối kép
-

1. Danh sách

- Thao tác trên danh sách tuyến tính
 - Khởi tạo danh sách (create)
 - Kiểm tra danh sách rỗng (isEmpty)
 - Kiểm tra danh sách đầy (isFull)
 - Tính kích thước (sizeOf)
 - Xóa rỗng danh sách (clear) *cuu duong than cong . com*
 - Thêm một phần tử vào danh sách tại một vị trí cụ thể (insert)
 - Loại bỏ một phần tử tại một vị trí cụ thể khỏi danh sách (remove)
 - Lấy một phần tử tại một vị trí cụ thể (retrieve)
 - Thay thế giá trị của một phần tử tại một vị trí cụ thể (replace)
 - Duyệt danh sách và thực hiện một thao tác tại các vị trí trong danh sách (traverse)
 -

1.1. Danh sách kế tiếp

- Sử dụng một vector lưu trữ gồm một số các ô nhớ liên tiếp để lưu trữ một danh sách tuyến tính
 - Các phần tử liền kề nhau được lưu trữ trong những ô nhớ liền kề nhau
 - Mỗi phần tử của danh sách cũng được gán một chỉ số chỉ thứ tự được lưu trữ trong vector
 - Tham chiếu đến các phần tử sử dụng địa chỉ được tính giống như lưu trữ mảng.

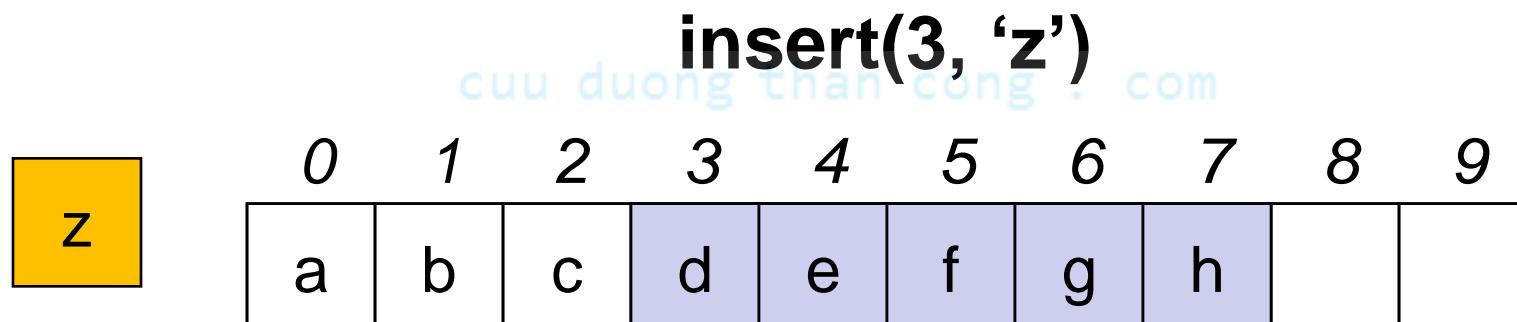


1.1. Danh sách kế tiếp

- **Ưu điểm** của cách lưu trữ kế tiếp
 - Tốc độ truy cập vào các phần tử của danh sách nhanh
- **Nhược điểm** của cách lưu trữ kế tiếp
 - Cần phải biết trước kích thước tối đa của danh sách
 - Tại sao?
 - Thực hiện các phép toán bổ sung các phần tử mới và loại bỏ các phần tử cũ khá tốn kém
 - Tại sao?

1.1.a. Thêm một phần tử vào một danh sách kế tiếp

- 2 trường hợp
 - `insert(index, element)`: thêm một phần tử `element` vào một vị trí cụ thể `index`
 - `insert(list, element)`: thêm một phần tử `element` vào vị trí bất kỳ trong danh sách `list`
- Điều kiện tiên quyết:
 - Danh sách phải được khởi tạo rồi
 - Danh sách chưa đầy cuuduongthancong.com
 - Phần tử thêm vào chưa có trong danh sách
- Điều kiện hậu nghiệm:
 - Phần tử cần thêm vào có trong danh sách



1.1.a. Thêm một phần tử vào một danh sách kế tiếp

Algorithm Insert

Input: index là vị trí cần thêm vào, element là giá trị cần thêm vào

Output: tình trạng danh sách

if list đầy

return overflow

if index nằm ngoài khoảng [0..count]

return range_error

//Dời tất cả các phần tử từ index về sau 1 vị trí

for i = count-1 **down to** index

 entry[i+1] = entry[i]

entry[index] = element // Gán element vào vị trí index

count++ // Tăng số phần tử lên 1

return success;

End Insert

1.1.b.Xóa 1 phần tử khỏi danh sách kế tiếp

remove(3, 'd')

0	1	2	3	4	5	6	7	8	9
a	b	c	d	e	f	g	h		

count=8

1.1.b.Xóa 1 phần tử khỏi danh sách kế tiếp

Algorithm Remove

Input: index là vị trí cần xóa bỏ, element là giá trị lấy ra được
Output: danh sách đã xóa bỏ phần tử tại index

```
if list rỗng
    return underflow
if index nằm ngoài khoảng [0..count-1]
    return range_error
element = entry[index]          //Lấy element tại vị trí index ra
count--                          //Giảm số phần tử đi 1
//Dời tất cả các phần tử từ index về trước 1 vị trí
for i = index to count-1
    entry[i] = entry[i+1]
return success;
```

End Remove

1.1.c.Duyệt danh sách kế tiếp

Algorithm Traverse

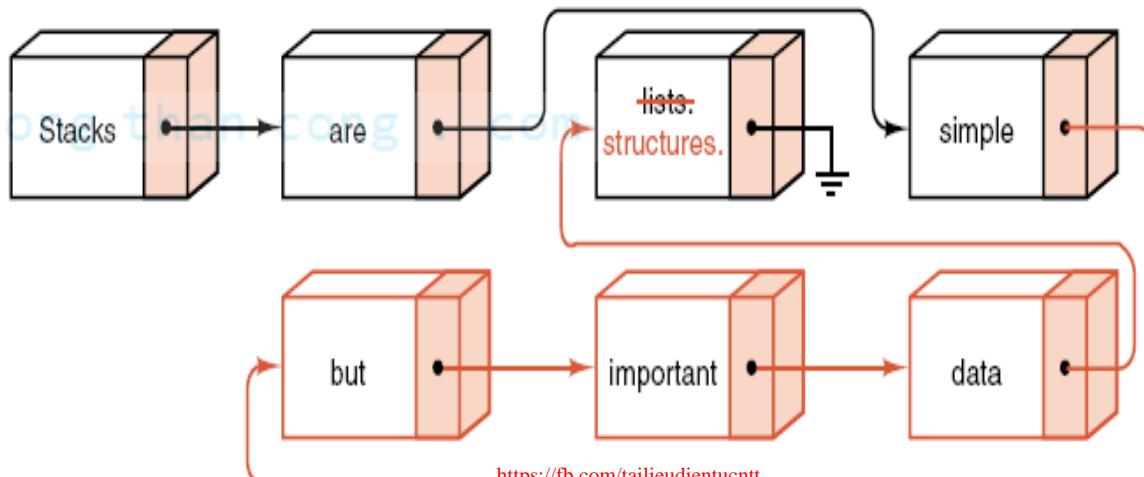
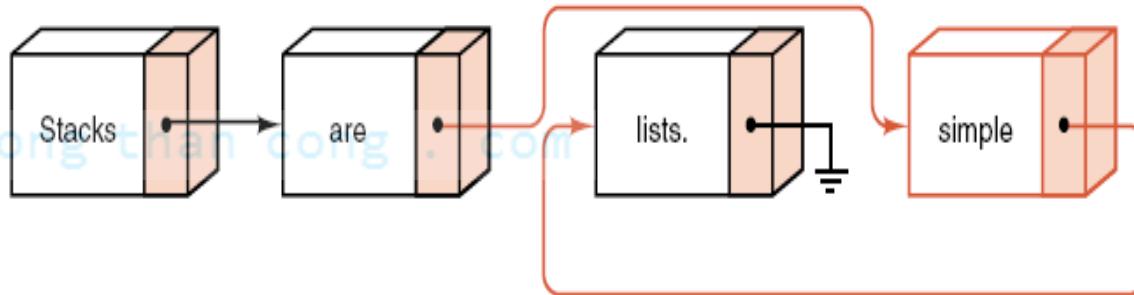
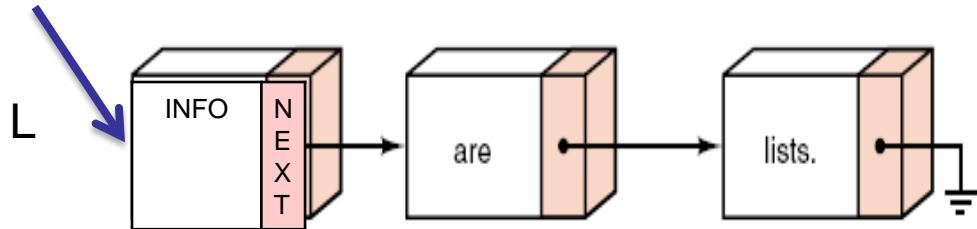
Input: hàm visit dùng để tác động vào từng phần tử
Output: danh sách được cập nhật bằng hàm visit

```
cuu duong than cong . com
//Quét qua tất cả các phần tử trong list
for index = 0 to count-1
    Thi hành hàm visit để duyệt phần tử entry[index]
```

End Traverse

1.2. Danh sách nối đơn

- Một phần tử trong danh sách = một nút
- Quy cách của một nút
 - INFO: chứa thông tin (nội dung, giá trị) ứng với phân tử
 - NEXT: chứa địa chỉ của nút tiếp theo
- Để thao tác được trên danh sách, cần nắm được địa chỉ của nút đầu tiên trong danh sách, tức là biết được con trỏ L trỏ tới đầu danh sách



Tổ chức danh sách mốc nối

- Nút = dữ liệu + mốc nối □

- Định nghĩa:

```
typedef struct node {
```

```
    typed data;
```

```
    struct node *next; } Node;
```

- Tạo nút mới:

```
Node *p = malloc(sizeof(Node)); □
```

- Giải phóng nút:

```
free(p);
```

Khởi tạo và truy cập danh sách móc nối

- Khai báo một con trỏ
Node *Head;

Head là con trỏ trỏ đến nút đầu của danh sách.Khi danh sách rỗng thì Head =NULL.

- Tham chiếu đến các thành phần của một nút trỏ bởi p
 - INFO(p)
 - NEXT(p)
- Một số thao tác với danh sách nối đơn
 - 1.Thêm một nút mới tại vị trí cụ thể
 - 2.Tìm nút có giá trị cho trước
 - 3.Xóa một nút có giá trị cho trước
 - 4.Ghép 2 danh sách nối đơn
 - 5.Hủy danh sách nối đơn

Truyền danh sách móc nối vào hàm

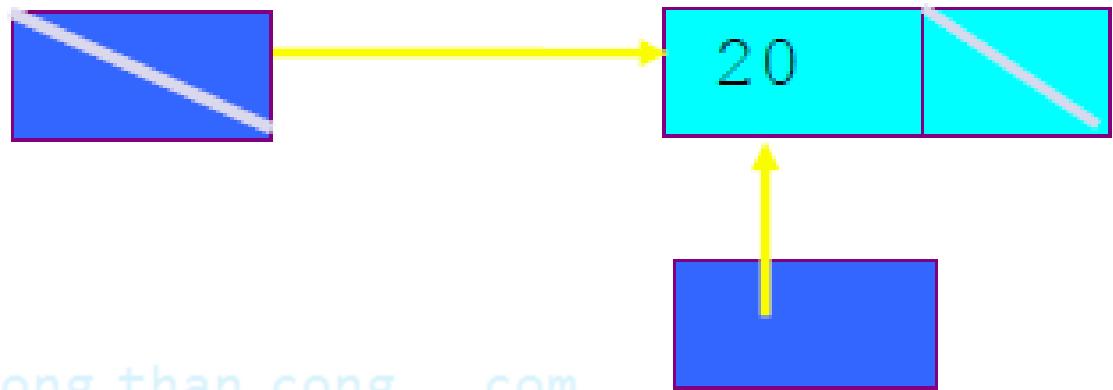
- ❑ Khi truyền danh sách móc nối vào hàm, chỉ cần truyền Head.
- ❑ Sử dụng Head để truy cập toàn bộ danh sách
 - ❑ Note: nếu hàm thay đổi vị trí nút đầu của danh sách (thêm hoặc xóa nút đầu) thì Head sẽ không còn trỏ đến đầu danh sách
 - ❑ Do đó nên truyền Head theo tham biến (hoặc trả lại một con trỏ mới)

Thêm một nút mới

- Các trường hợp của thêm nút
 - 1.Thêm vào danh sách rỗng
 - 2.Thêm vào đầu danh sách
 - 3.Thêm vào cuối danh sách . com
 - 4.Thêm vào giữa danh sách
- Thực tế chỉ cần xét 2 trường hợp
 - Thêm vào đầu danh sách(TH1 và TH2)
 - Thêm vào giữa hoặc cuối danh sách(TH3 và TH4)

Thêm vào danh sách rỗng

- Head = NULL



```
Node *newNode;  
newNode=  
    malloc(sizeof(Node));  
newNode->data = 20;  
newNode->next = NULL;  
Head = newNode;
```

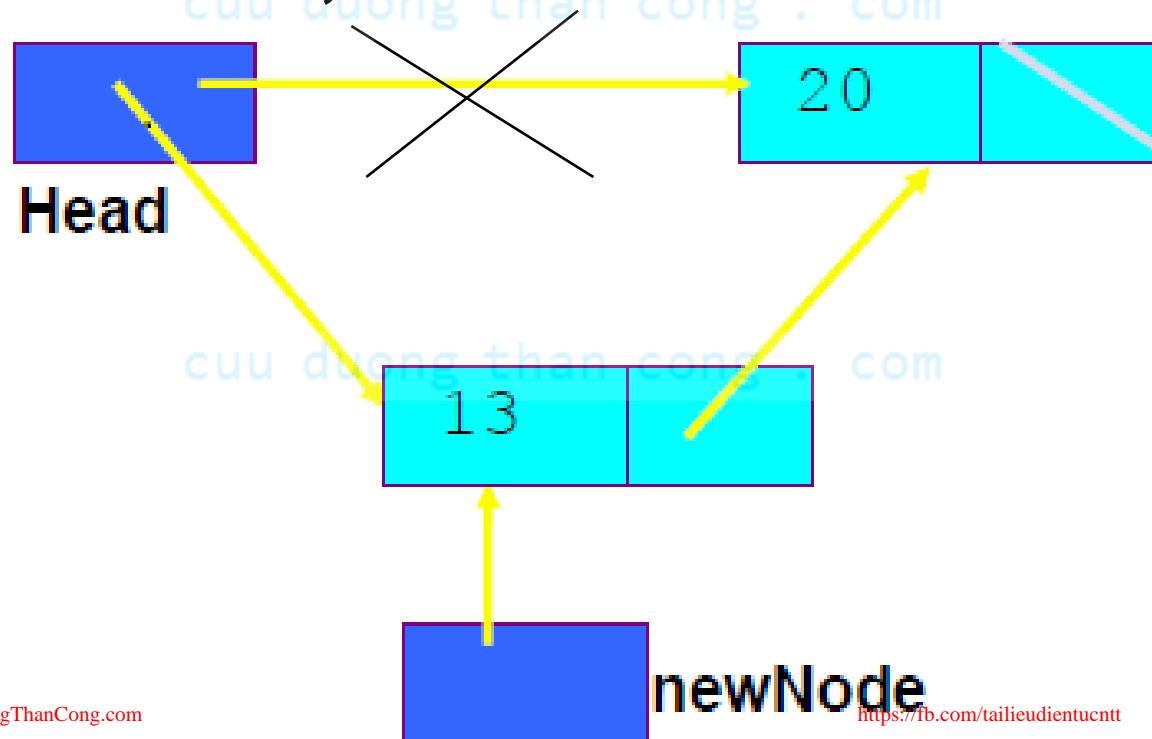
Thêm một nút vào đầu danh sách

```
newNode= malloc(sizeof(Node));
```

```
newNode->data = 13;
```

```
newNode->next = Head;
```

```
Head = newNode;
```



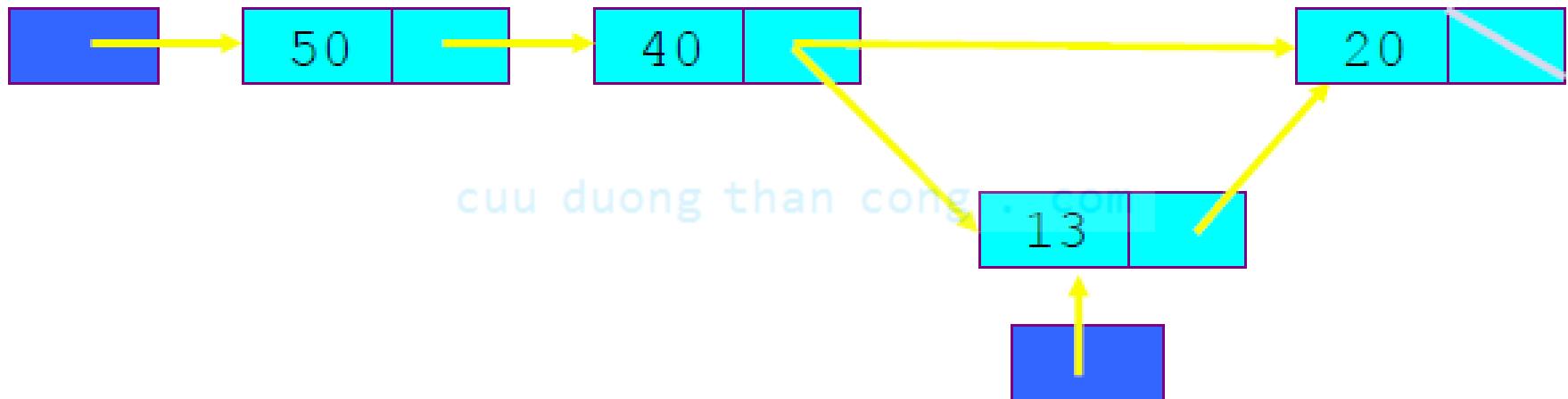
Thêm một nút vào giữa/cuối danh sách

```
newNode= malloc(sizeof(Node));
```

```
newNode->data = 13;
```

```
newNode->next = currNode->next;
```

```
currNode->next= newNode;
```



Thêm một nút mới

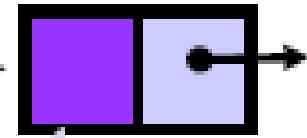
- **Node *InsertNode(Node *head, int index, int x)**
 - Thêm một nút mới với dữ liệu là x vào sau nút thứ index.(ví dụ,khi index = 0, nút được thêm là phần tử đầu danh sách;khi index = 1, chèn nút mới vào sau nút đầu tiên,v.v)
 - Nếu thao tác thêm thành công,trả lại nút được thêm. Ngược lại,trả lạiNULL.
 - (Nếu index < 0 hoặc > độ dài của danh sách,không thêm được.)
- Giải thuật

1.Tìm nút thứ index –currNode

2.Tạo nút mới

3.Móc nối nút mới vào danh sách

currNode



newNode->next = currNode->next;

currNode->next = newNode;

SE-SolCT

05/04/2019

Last Update 8-2010
CuuDuongThanCong.com

<https://fb.com/tailieudientucntt>

Thêm một nút mới

```
Node * InsertNode(Node *head,int index,int x)
{
    if (index < 0) return NULL;
    int currIndex = 1;
    Node *currNode = head;
    while(currNode && index > currIndex) {
        currNode = currNode->next;
        currIndex++;
    }
    if (index < 0 && currNode== NULL) return NULL;
    Node *newNode = (Node *) malloc(sizeof(Node));
    newNode->data = x;
    if (index == 0) {
        newNode->next = head;
        head = newNode;
    }
    else {
        newNode->next = currNode->next;
        currNode->next = newNode;
    }
    return newNode;
}
```

Tìm nút thứ index, nếu
Không tìm được trả về
NULL

Tạo nút mới

Thêm vào đầu ds

Thêm vào sau currNode

Tìm nút

- **int FindNode(int x)**
 - □ Tìm nút có giá trị x trong danh sách.
 - □ Nếu tìm được trả lại vị trí của nút.Nếu không, trả lại 0.

```
Int FindNode(Node *head,int x) {  
    Node *currNode = head;  
    int currIndex = 1;  
    while (currNode && currNode->data != x) {  
        currNode = currNode->next;  
        currIndex++;  
    }  
    if (currNode) return currIndex;  
    else return 0;
```

Xóa nút

- int DeleteNode(int x)
 - □ Xóa nút có giá trị bằng x trong danh sách.
 - □ Nếu tìm thấy nút, trả lại vị trí của nó.
Nếu không, trả lại 0.
 - □ Giải thuật
 - Tìm nút có giá trị x (tương tự như FindNode)
 - Thiết lập nút trước của nút cần xóa nối đến nút sau của nút cần xóa
 - Giải phóng bộ nhớ cấp phát cho nút cần xóa
 - Giống như InsertNode, có 2 trường hợp
 - Nút cần xóa là nút đầu tiên của danh sách
 - Nút cần xóa nằm ở giữa hoặc cuối danh sách

```

Int DeleteNode(Node *head, int x) {
    Node *prevNode = NULL;
    Node *currNode = head;
    int currIndex = 1;
    while (currNode && currNode->data != x) {
        prevNode = currNode;
        currNode = currNode->next;
        currIndex++;
    }
    if (currNode) { cuu duong than cong . com
        if (prevNode) {
            prevNode->next = currNode->next;
            free (currNode);
        } else {
            head = currNode->next;
            free (currNode); than cong . com
        }
    }
    return currIndex;
}
return 0;

```

Tìm nút có giá trị = x

Xóa nút ở giữa

Xóa nút head

Hủy danh sách

- void DestroyList(Node *head)
- □ Dùng để giải phóng bộ nhớ được cấp phát cho danh sách.
- □ Duyệt toàn bộ danh sách và xóa lần lượt từng nút.

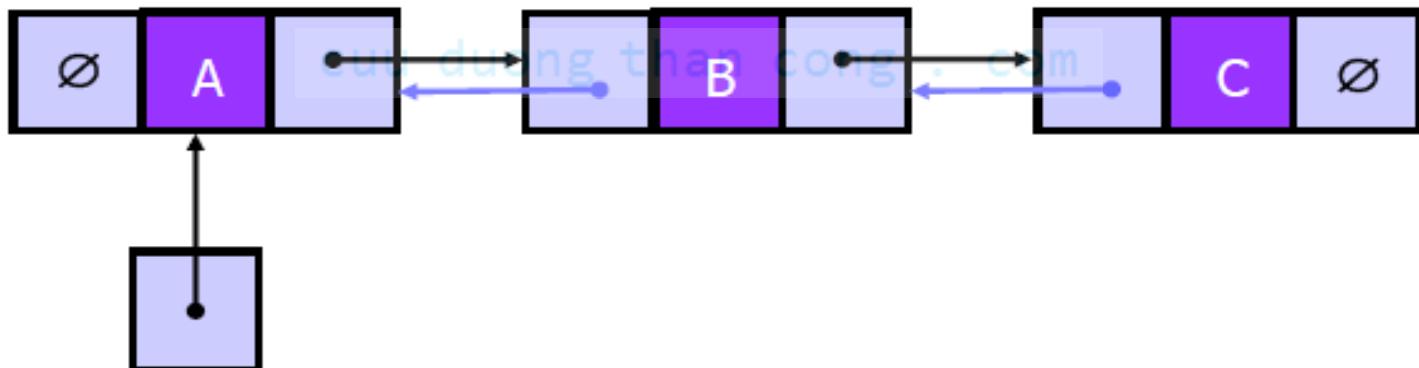
```
Void DestroyList(Node* head){  
    Node *currNode = head, *nextNode= NULL;  
    while(currNode != NULL){  
        nextNode = currNode->next;  
        free(currNode); // giải phóng nút vừa duyệt  
        currNode = nextNode;  
    }  
}
```

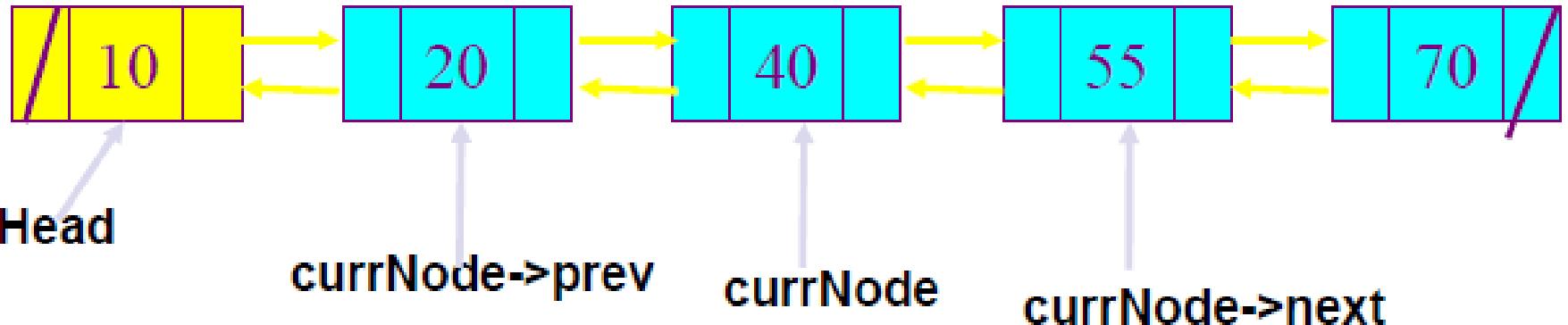
So sánh mảng và danh sách liên kết

- Việc lập trình và quản lý danh sách liên kết khó hơn mảng, nhưng nó có những ưu điểm:
- **Linh động**: danh sách liên kết có kích thước tăng hoặc giảm rất linh động.
 - Không cần biết trước có bao nhiêu nút trong danh sách. Tạo nút mới khi cần.
 - Ngược lại, kích thước của mảng là cố định tại thời gian biên dịch chương trình.
- **Thao tác thêm và xóa dễ dàng**
 - Để thêm và xóa một phần tử mảng, cần phải copy dịch chuyển phần tử.
 - Với danh sách móc nối, không cần dịch chuyển mà chỉ cần thay đổi các móc nối

Danh sách nối kép

- Mỗi nút không chỉ nối đến nút tiếp theo mà còn nối đến nút trước nó
 - Có 2 mối nối NULL: tại nút đầu và nút cuối của danh sách
 - Ưu điểm: tại một nút có thể thăm nút trước nó một cách dễ dàng. Cho phép duyệt danh sách theo chiều ngược lại





- Mỗi nút có 2 mối nối
 - prev nối đến phần tử trước
 - next nối đến phần tử sau

```
typedef struct Node{
    int data;          cuu duong than cong . com
    structNode *next;
    structNode *prev;
} Node;
```

95/04/2010
Last Update: 08/2010
CuuDuongThanCong.com

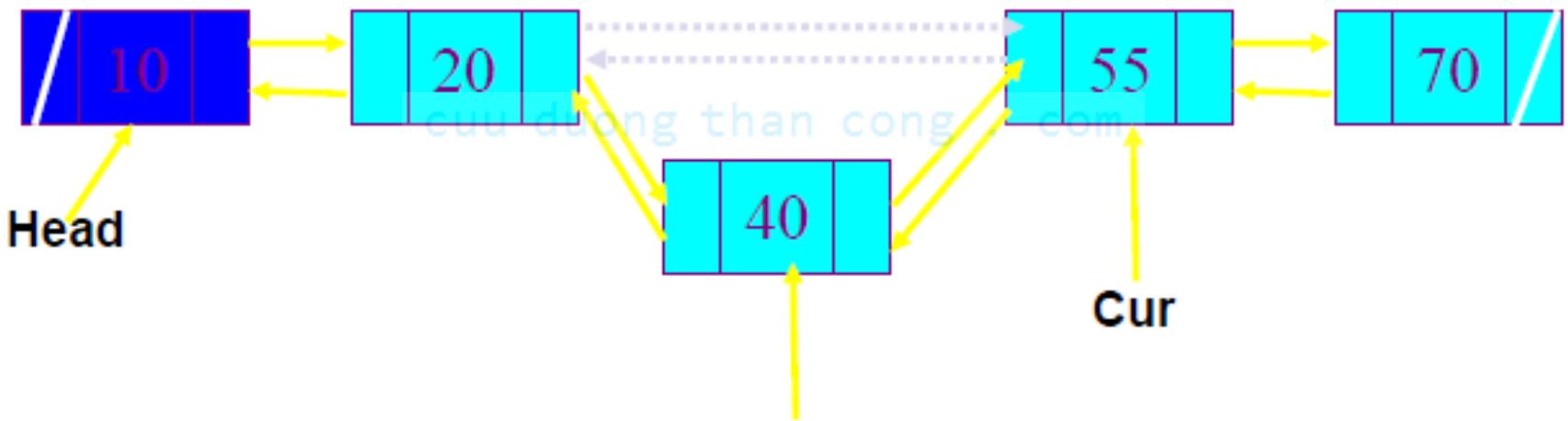
- Thêm nút New nằm ngay trước Cur (không phải nút đầu hoặc cuối danh sách)

New->next = Cur;

New->prev= Cur->prev;

Cur->prev= New;

(New->prev)->next = New;



- Xóa nút Cur(không phải nút đầu hoặc cuối danh sách)

(Cur->prev)->next = Cur->next;

(Cur->next)->prev = Cur->prev;

free (Cur);

05/04/2019

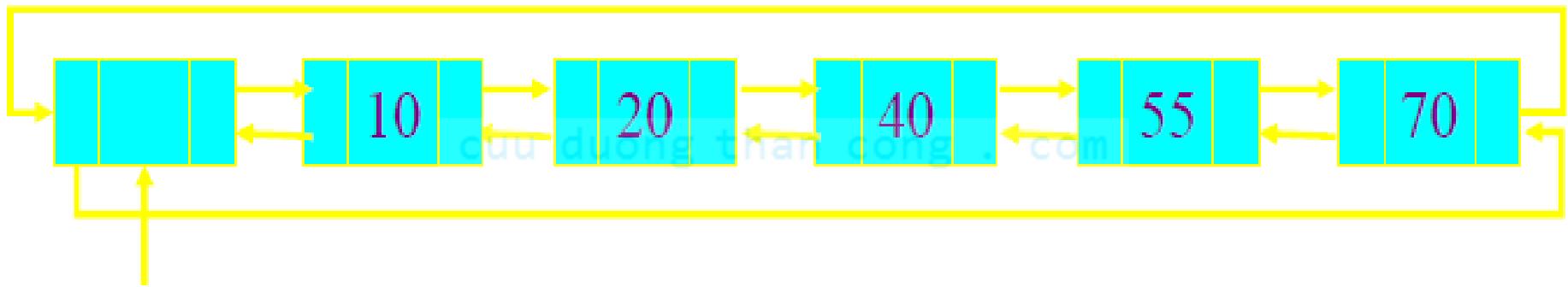
Last Update 8-2010

SE-SolICT

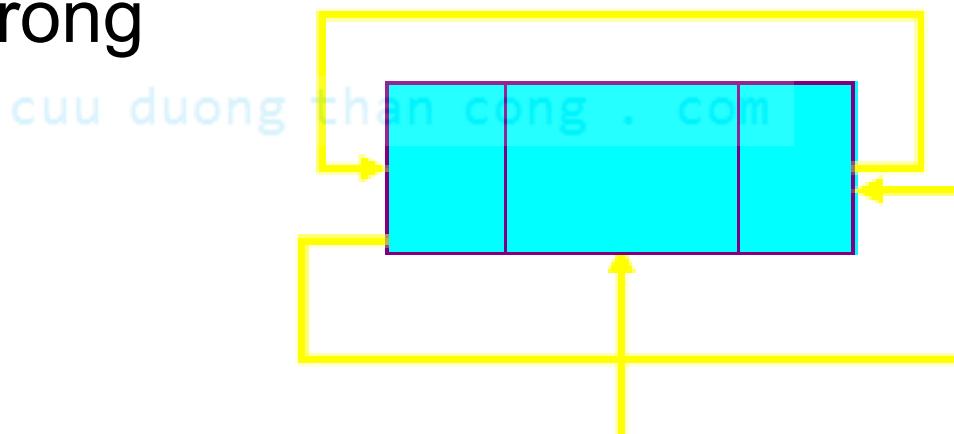
KT LT4-2.32

Danh sách nối kép với nút đầu giả

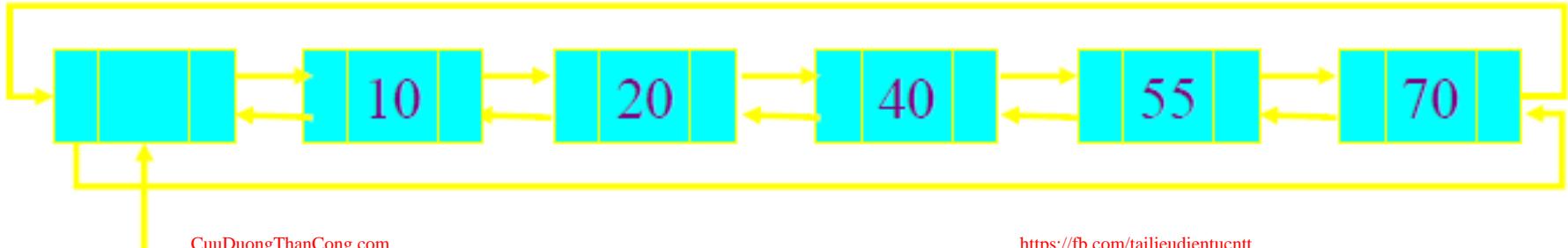
- Danh sách không rỗng



- Danh sách rỗng



- Tạo danh sách nối kép rỗng
Node *Head = malloc (sizeof(Node));
Head->next = Head;
Head->prev = Head;
- Khi thêm hoặc xóa các nút tại đầu ds, giữa hay cuối ds ???



Xóa nút

```
void deleteNode(Node *Head, int x){  
    Node *Cur;  
    Cur = FindNode(Head, x);  
    if (Cur != NULL){  
        Cur->prev->next = Cur->next;  
        Cur->next->prev= Cur->prev;  
        free(Cur);  
    }  
}
```

Thêm nút

```
void insertNode(Node *Head, int item) {  
    Node *New, *Cur;  
    New = malloc(sizeof(Node));  
    New->data = item;  
    Cur = Head->next;  
    while (Cur != Head){  
        if (Cur->data < item)  
            Cur = Cur->next;  
        else  
            break;  
    }  
    New->next = Cur;uu duong than cong . com  
    New->prev= Cur->prev;  
    Cur->prev= New;  
    (New->prev)->next = New;  
}
```

Bài tập

Sử dụng danh sách mốc nối kép với nút đầu giả, xây dựng bài toán quản lý điểm SV đơn giản, với các chức năng sau :

1. Nhập dữ liệu vào ds
2. Hiện thị dữ liệu 1 lớp theo thứ tự họ và tên
3. Sắp xếp dữ liệu
4. Tìm kiếm kết quả

c định nghĩa trong cấu trúc sau

```
typedef struct {
```

```
    int masv; // mã hiệu sv
```

```
    char malop[12];  
    char hovaten[30];
```

```
    float diemk1;
```

```
    float diemk2;
```

```
};
```

05/04/2010
Last Update: 8-2010
conduongthancong.com

SE-SoICT

KT LT4-2.37

1.3 Ngăn xếp và hàng đợi

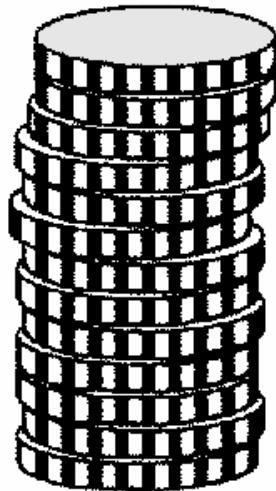
- 1. Định nghĩa Stack
- 2. Lưu trữ kế tiếp với Stack (sử dụng mảng)
- 3. Ứng dụng của Stack *cuu_duong_than_cong . com*
- 4. Định nghĩa Queue
- 5. Lưu trữ kế tiếp với Queue (sử dụng mảng)
- 6. Ứng dụng của Queue
- 7. Lưu trữ mốc nối với Stack *cuu_duong_than_cong . com*
- 8. Lưu trữ mốc nối với Queue (bài tập)

1. Định nghĩa Stack

- **i danh sách tuyến tính đặc biệt:**
 - Ngăn xếp –Stack
 - Hàng đợi –Queue
-

LIFO.

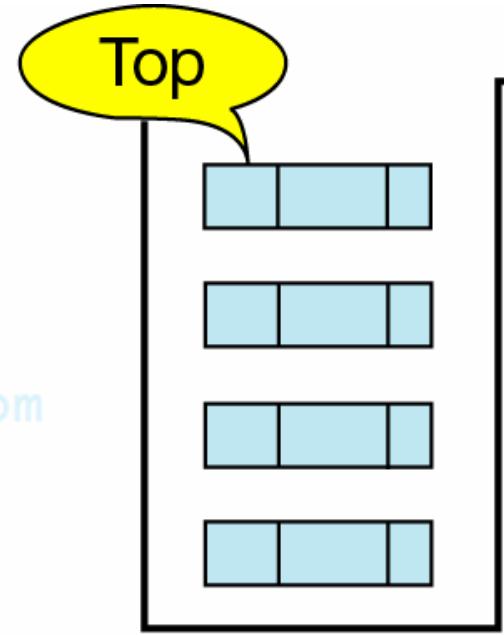
Stack trong thực tế



Stack of coins



Stack of books



Computer stack

Stack là một cấu trúc LIFO: Last In First Out

05/04/2019

SE-SolCT

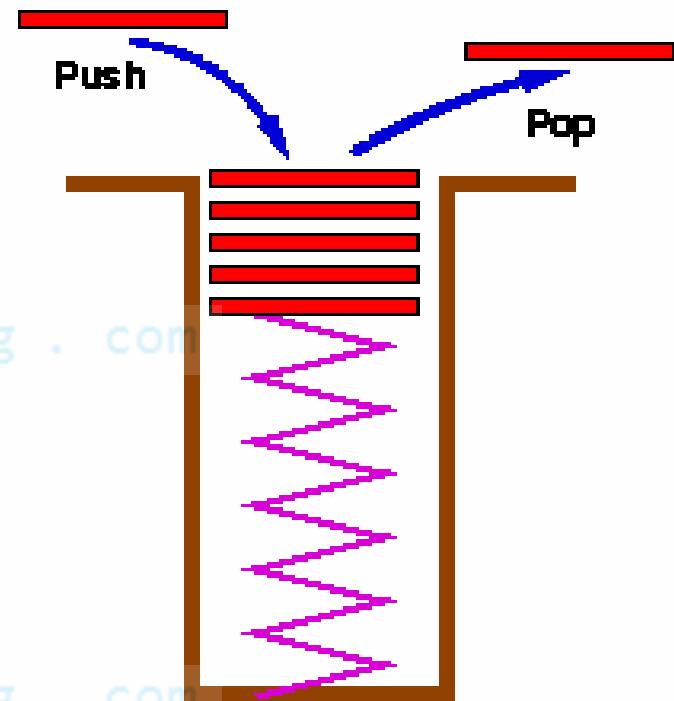
KTE14-2.40

Last Update 8-2010
CuuDuongThanCong.com

<https://fb.com/tailieudientucntt>

Các thao tác cơ bản trên Stack

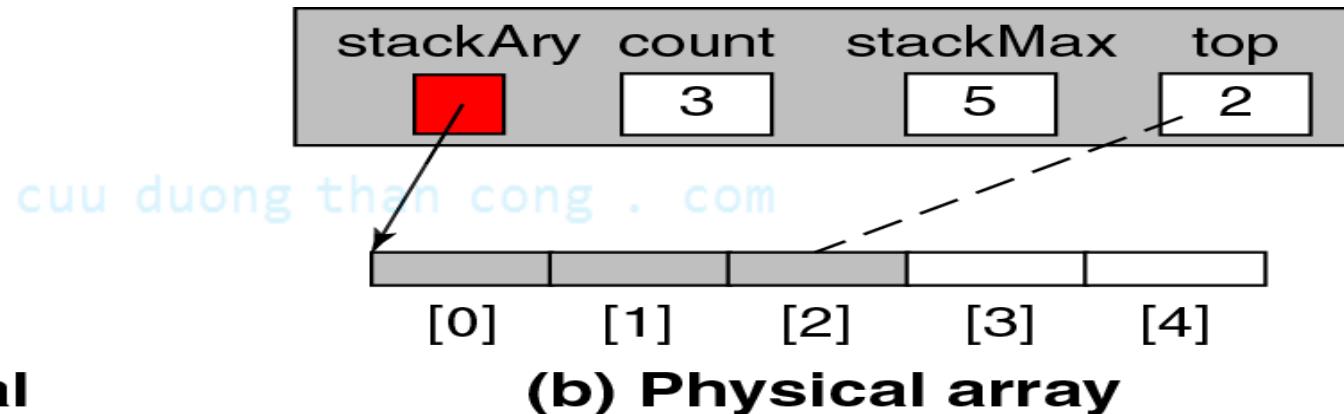
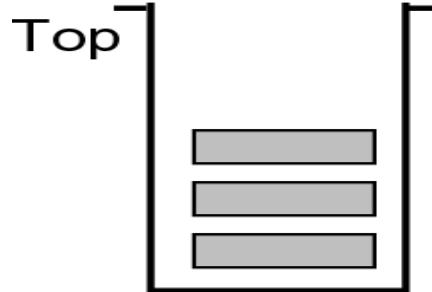
- Push
 - Thêm một phần tử
Tràn (overflow)
- Pop
 - Xóa một phần tử
Underflow
- Top
 - Phần tử đỉnh
stack rỗng
- Kiểm tra rỗng/đầy



Lưu trữ Stack

2 cách lưu trữ:

- Lưu trữ kế tiếp: sử dụng mảng



Cấu trúc dữ liệu

```
/* Stack của các số nguyên: intstack*/
typedef struct intstack{
    Int *stackArr; /*mảng lưu trữ các phần tử*/
    Int count;      /*số ptử hiện có của stack */
    Int stackMax;  /* giới hạn Max của số ptử*/
    Int top;        /*chỉ số của phần tử đỉnh*/
} IntStack;
```

Tạo Stack

```
IntStack *CreateStack(int max){  
    IntStack *stack;  
    stack =(IntStack *) malloc(sizeof(IntStack));  
    if (stack == NULL)  
        return NULL;  
    /*Khởi tạo stack rỗng*/  
    stack->top = -1;  
    stack->count = 0;  
    stack->stackMax= max;  
    stack->stackArr=malloc(max * sizeof(int));  
    return stack ;  
}
```

Push

```
Int PushStack(IntStack *stack, int dataIn) {  
    /*Kiểm tra tràn*/  
    if(stack->count == stack->stackMax)  
        Return 0;  
    /* Thêm phần tử vào stack */  
    (stack->count)++;  
    (stack->top)++; /* Tăng đỉnh */  
    stack->stackArr[stack->top] =dataIn;  
  
    Return 1;  
}
```

Pop

```
Int PopStack(IntStack *stack, int *dataOut){  
    /* Kiểm tra stack rỗng */  
    if(stack->count == 0)  
        return 0;  
    /* Lấy giá trị phần tử bị loại */  
    *dataOut=stack->stackArr[stack->top];  
    (stack->count)--;  
    (stack->top)--; /* Giảm đỉnh */  
    Return 1;  
}
```

Top

```
Int TopStack(IntStack *stack, int *dataOut){  
    if(stack->count == 0) // Stack rỗng  
        return 0;  
    *dataOut = stack->stackArr[stack->top];  
    return 1;  
}
```

Kiểm tra rỗng ?

```
Int IsEmptyStack(IntStack *stack){  
    return(stack->count == 0);  
}
```

Kiểm tra đầy ?

```
int IsFullStack(IntStack *stack) {  
    return(stack->count == stack->stackMax);  
}
```

Ứng dụng của Stack

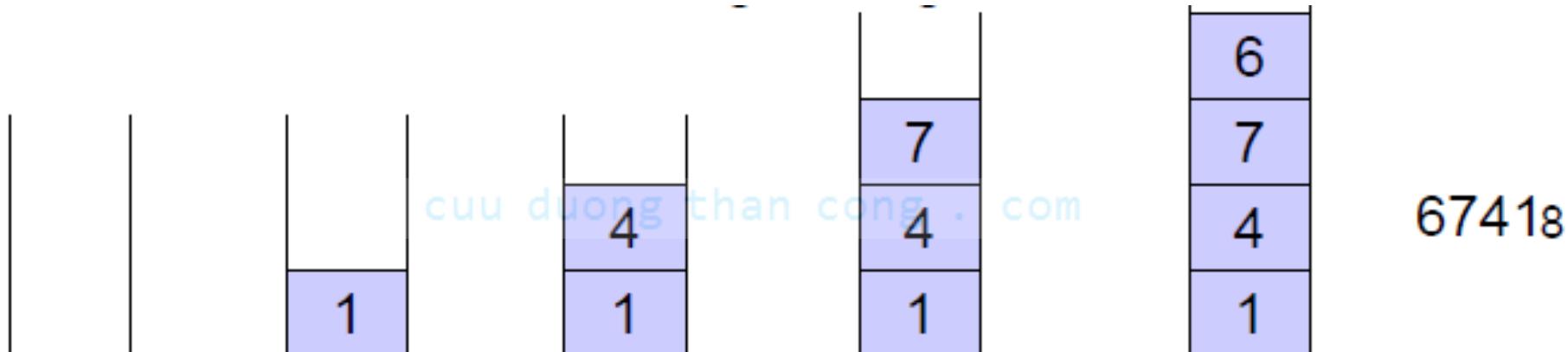
- Bài toán đổi cơ số: Chuyển một số từ hệ thập phân sang hệ cơ số bất kỳ
 - (base 8) $28_{10} = 3 \cdot 8^1 + 4 \cdot 8^0 = 34_8$
 - (base 4) $72_{10} = 1 \cdot 4^3 + 0 \cdot 4^2 + 2 \cdot 4^1 + 0 \cdot 4^0 = 1020_4$
 - (base 2) $53_{10} = 1 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 110101_2$

Đầu vào số thập phân n , cơ số a

Đầu ra số hệ cơ số b tương đương

1. Chữ số bên phải nhất của kết quả = $n \% b$. Đẩy vào Stack.
2. Thay $n = n / b$ (để tìm các số tiếp theo).
3. Lặp lại bước 1-2 cho đến khi $n = 0$.
4. Rút lần lượt các chữ số lưu trong Stack, chuyển sang dạng ký tự tương ứng với hệ cơ số trước khi in ra kết quả

Ví dụ : Đổi 3553 ($a = 10$) sang cơ số $b = 8$



Stack rỗng
 $n = 3553$

$$\begin{aligned}n \% 8 &= 1 \\n / 8 &= 444 \\n &= 444\end{aligned}$$

$$\begin{aligned}n \% 8 &= 4 \\n / 8 &= 55 \\n &= 55\end{aligned}$$

$$\begin{aligned}n \% 8 &= 7 \\n / 8 &= 6 \\n &= 6\end{aligned}$$

$$\begin{aligned}n \% 8 &= 6 \\n / 8 &= 0 \\n &= 0\end{aligned}$$

Chuyển sang dạng ký tự tương ứng:

```
Char *digitChar= “0123456789ABCDEF”;  
char d = digitChar[13]; // 1310= D16  
char f = digitChar[15]; // 1510= F16
```

cuu duong than cong . com

Đổi cơ số

```
void DoiCoSo(int n,int b) {  
    char*digitChar= "0123456789ABCDEF“;  
    //Tạo một stack lưu trữ kết quả  
    IntStack *stack = CreateStack(MAX);  
    do{  
        //Tính chữ số bên phải nhất, đẩy vào stack  
        PushStack(stack, n% b);  
        n/= b; //Thay n = n/b để tính tiếp  
    }while(n!= 0); //Lặp đến khi n = 0  
    while( !IsEmptyStack(stack ){  
        // Rút lần lượt từng phần tử của stack  
        PopStack(stack, &n);  
        // chuyển sang dạng ký tự và in kết quả  
        printf("%c", digitChar[n]);  
    }  
}
```

Ứng dụng của Stack (tiếp)

Các biểu thức số học được biểu diễn bằng ký pháp trung tố . Với phép toán 2 ngôi: Mỗi toán tử được đặt giữa hai toán hạng. Với phép toán một ngôi: Toán tử được đặt trước toán hạng: vd

$$-2 + 3 * 5 \Leftrightarrow (-2) + (3 * 5)$$

- Thứ tự ưu tiên của các phép tử:

$$() > ^ > * = \% = / > + = -$$

- Việc đánh giá biểu thức trung tố khá phức tạp

Ký pháp hậu tố

Là giải pháp thay thế ký pháp trung tố, trong đó : Toán hạng đặt *trước* toán tử, Không cần dùng các dấu () .

Ví dụ :

$$a^*b^*c^*d^*e^*f \quad \text{cúu duong thanh} \Rightarrow ab^*c^*d^*e^*f^*$$

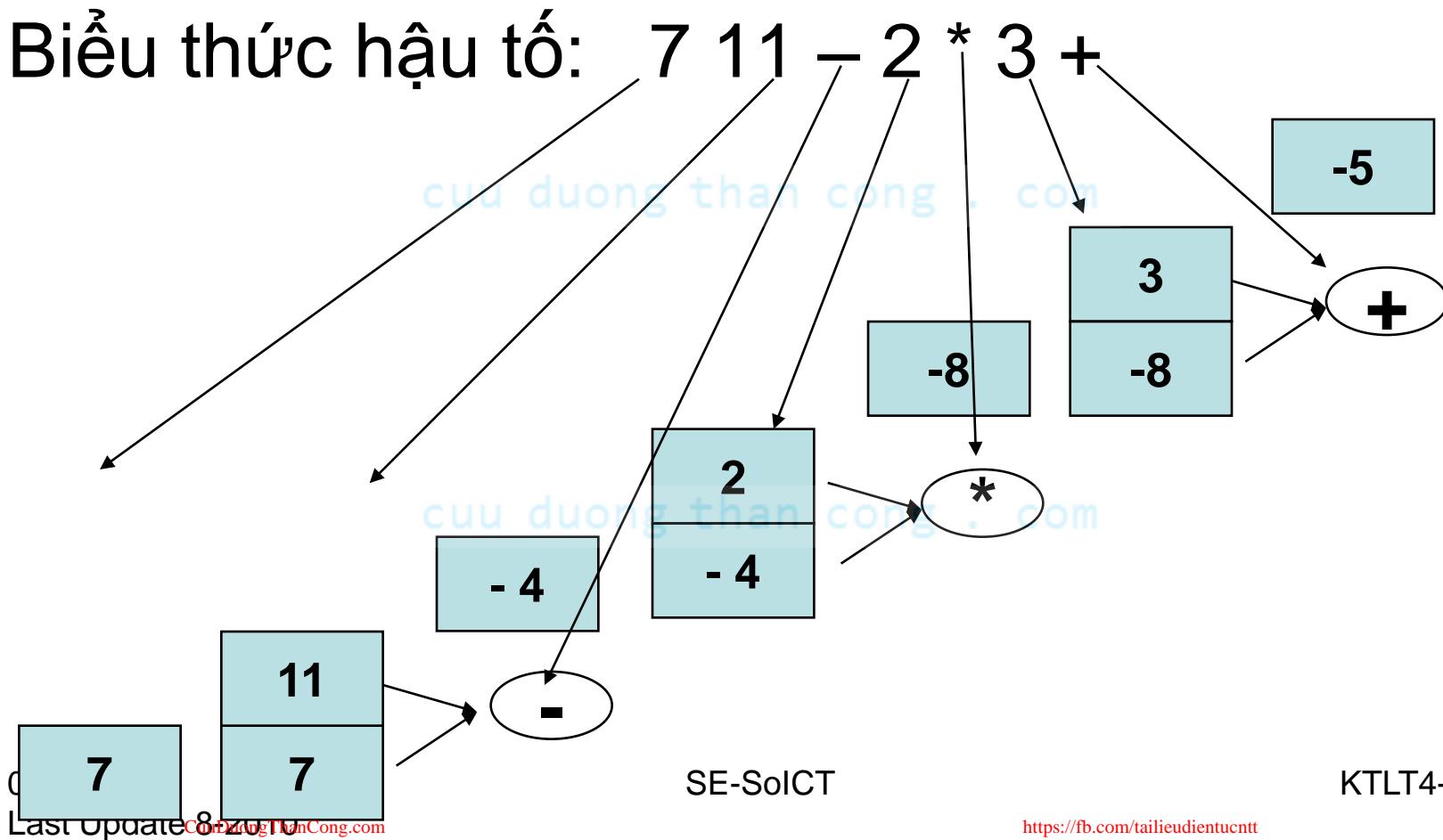
$$1 + (-5) / (6 * (7+8)) \Rightarrow 1\ 5\ -6\ 7\ 8\ +\ *\ / \ +$$

$$(x/y-a^*b) * ((b+x) -y) \Rightarrow x\ y\ / a\ b\ * -b\ x\ +\ y\ y\ ^{-*}$$

$$(x^*y^*z -x^2 / (y^*2 -z^3) + 1/z)^* (x -y) \Rightarrow \\ xy^*z^*x2^y2^z3^ -/ -1z/+xy -^*$$

Tính giá trị biểu thức hậu tố

Biểu thức trung tố: $(7 - 11) * 2 + 3$



Tính giá trị của biểu thức hậu tố

- Tính giá trị của một biểu thức hậu tố được lưu trong một xâu ký tự và trả về giá trị kết quả
- Với :
 - Toán hạng: Là các số nguyên không âm một chữ số (cho đơn giản)
 - Toán tử: + , - , * , / , % , ^

```
Bool isOperator(char op) {  
    return op == '+' || op == '-' ||  
           op == '*' || op == '%' ||  
           op == '/' || op == '^';  
}
```

```
Int compute(int left, int right, char op){  
    int value;  
    switch(op){  
        case '+': value = left + right; break;  
        case '-': value = left - right; break;  
        case '*': value = left * right; break;  
        case '%': value = left % right; break;  
        case '/': value = left / right; break;  
        case '^': value = pow(left, right);  
    }  
    return value;  
}
```

```

Int TinhBtHauTo(string Bt) {
    Int left, right, kq;
    char ch;
    IntStack *stack = CreateStack(MAX);
    for(int i=0; i < Bt.length(); i++)
    {
        ch = Bt[i];
        if ( isdigit(ch) )
            PushStack(stack, ch-'0'); // đẩy toán hạng vào stack
        else if (isOperator(ch)) {
            // rút stack 2 lần để lấy 2 toán hạng left và right
            PopStack(stack, &right);
            PopStack(stack, &left);
            kq =compute(left, right, ch); // Tính "left op right"
            PushStack(stack, kq); // Đẩy kq vào stack
        } else //không phải toán hạng hoặc toán tử
        printf("Bieu thuc loi");
    }
    // Kết thúc tính toán, giá trị biểu thức nằm trên đỉnh stack, đưa vào kq
    PopStack(stack, &kq);
    Return kq;
}

```

Bài tập

- Câu lệnh có hậu tố với các toán hạng tổng quát (có thể là số thực, có thể âm ...)
- Xây dựng chương trình chuyển

ưu tiên như sau : () > ^ > *

= % = / > + = -

cuuduongthancong . com

Queue

- Là danh sách mà thêm phải được thực hiện tại một đầu còn xóa phải thực hiện tại đầu kia.
- Queue là một kiểu cấu trúc FIFO: First In First Out



(a) A queue (line) of people



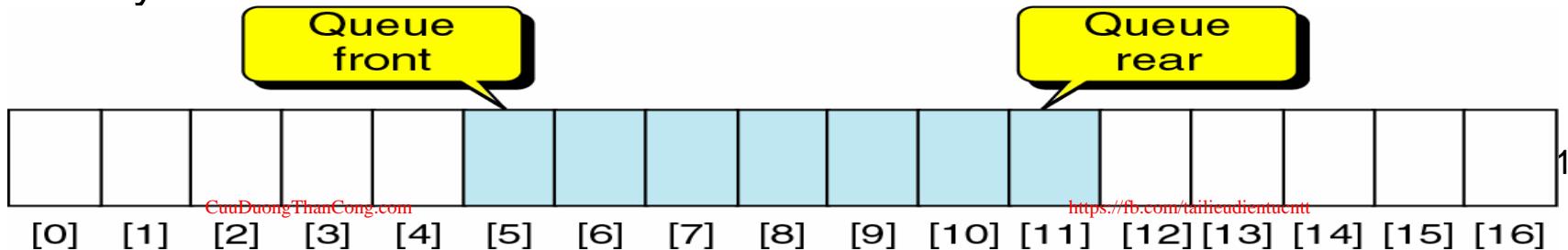
(b) A computer queue

- Phần tử đầu hàng sẽ được phục trước, phần tử này được gọi là **front**, hay **head** của hàng. Tương tự, phần tử cuối hàng , cũng là phần tử vừa được thêm vào hàng, được gọi là **rear** hay **tail** của hàng.

cuu duong than cong . com

Các phương án thực hiện hàng

- **Mô hình vật lý**
 - Có thể dùng 1 mảng. Tuy nhiên, cần phải nắm giữ cả **front** và **rear**.
 - Một cách đơn giản là ta luôn giữ front luôn là vị trí đầu của dãy. Lúc đó nếu thêm PT vào hàng ta chỉ việc thêm vào cuối dãy. Nhưng nếu lấy ra 1 pt ta phải dịch chuyển tất cả các pt của dãy lên 1 vị trí.
 - Mặc dù cách làm này rất giống với hình ảnh hàng đợi trong thực tế, nhưng lại là 1 lựa chọn rất dở với máy tính
- **Hiện thực tuyến**
 - Ta dùng 2 chỉ số Front và Rear để lưu trữ đầu và cuối hàng mà không di chuyển các phần tử.
 - Khi thêm ta chỉ việc tăng rear lên 1 và thêm pt vào vị trí đó
 - Khi rút pt ra, ta lấy pt tại front và tăng front lên 1
 - Nhược điểm : front và Rear chỉ tăng mà không giảm => lãng phí bộ nhớ
 - Có thể cải tiến bằng cách khi hàng đợi rỗng thì ta gán lại front=rear= đầu dãy



• Hiện thực của dãy vòng

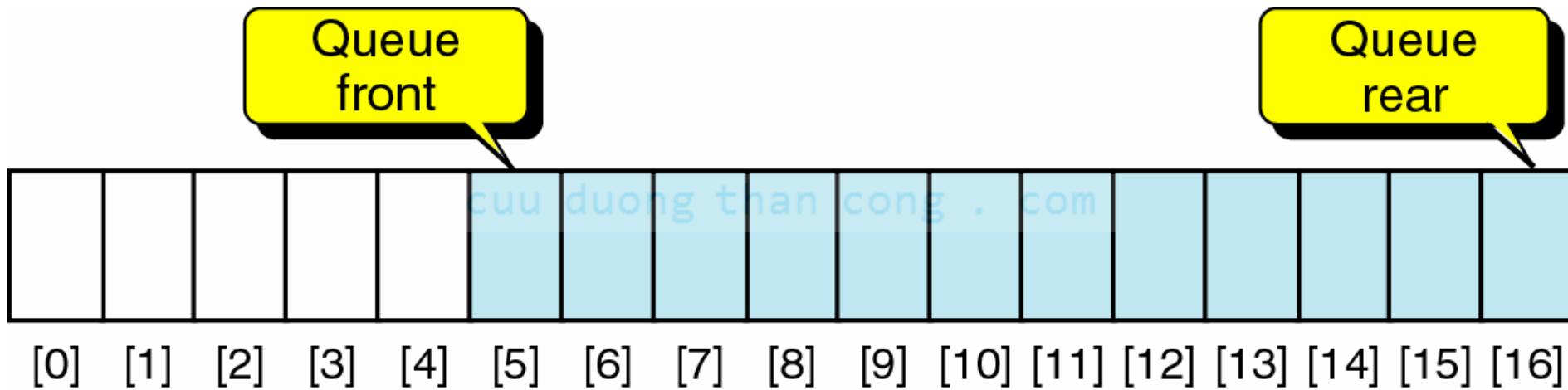
- Ta dùng 1 dãy tuyến tính để mô phỏng 1 dãy vòng.
- Các vị trí trong vòng tròn được đánh số từ 0 đến max-1, trong đó max là tông số PTỦ.
- Để thực hiện dãy vòng, chúng ta cũng sử dụng các phân tử được đánh số tương tự dãy tuyến tính.
- Sự thay đổi các chỉ số chỉ đơn giản là phép lấy phần dư số học: khi một chỉ số vượt quá max-1, nó sẽ bắt đầu trở lại với trị 0. Điều này tương tự với việc cộng thêm giờ trên đồng hồ mặt tròn

$i = ((i+1) == max) ? 0: (i+1);$

Hoặc if ((i+1) == max) i = 0; else i = i+1;

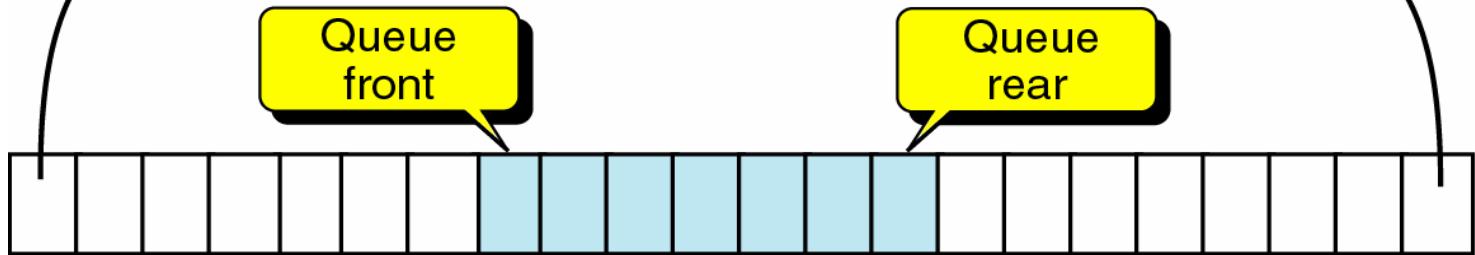
Hoặc $i = (i+1) \% max;$

Queue tăng hết mảng



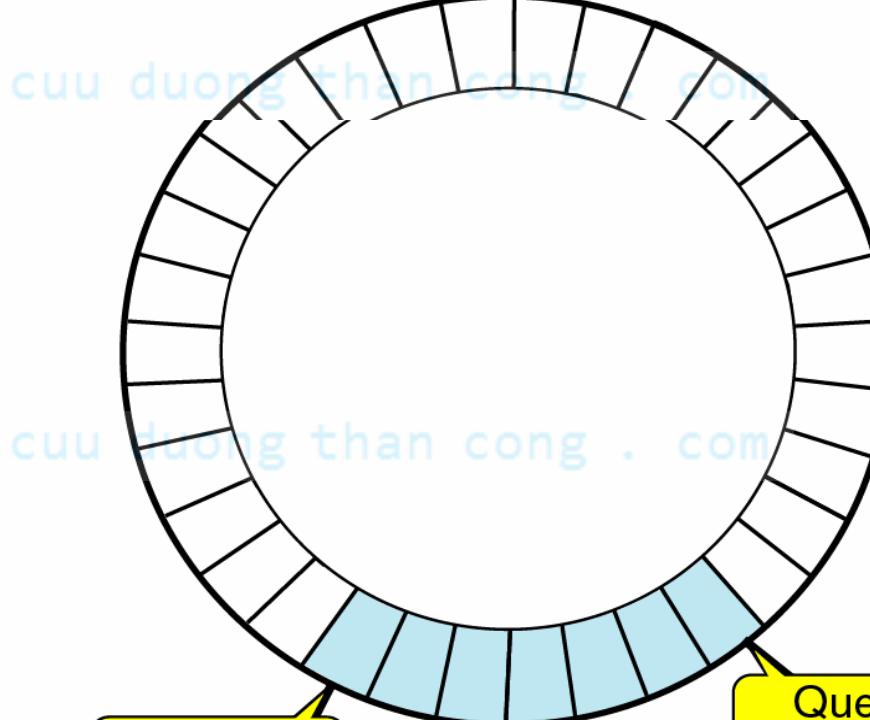
- Phải xử dụng mảng với kích thước lớn

Bend ends up to form circular queue.

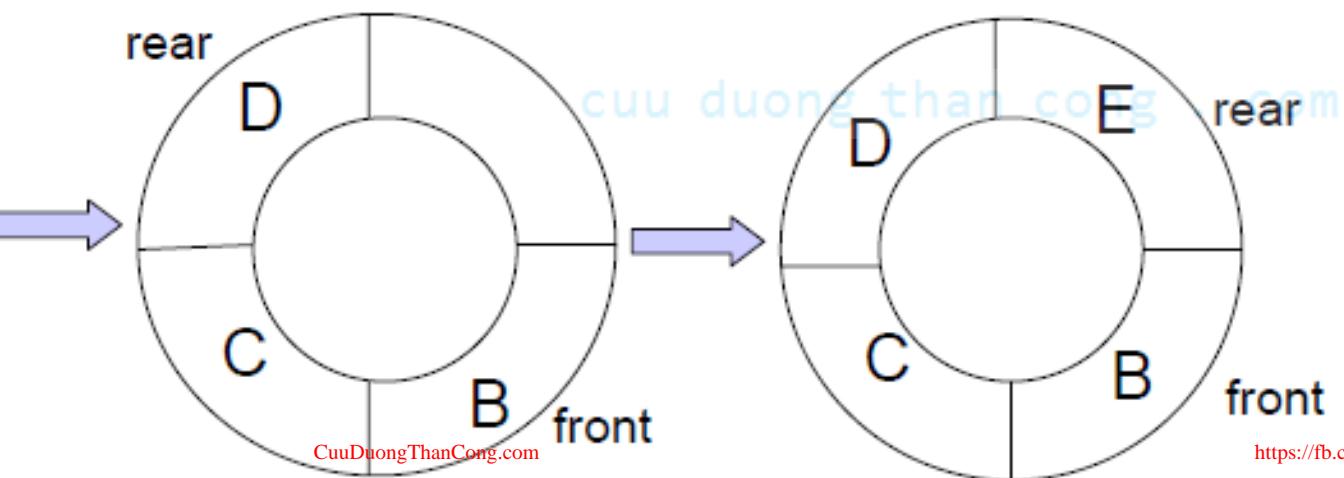
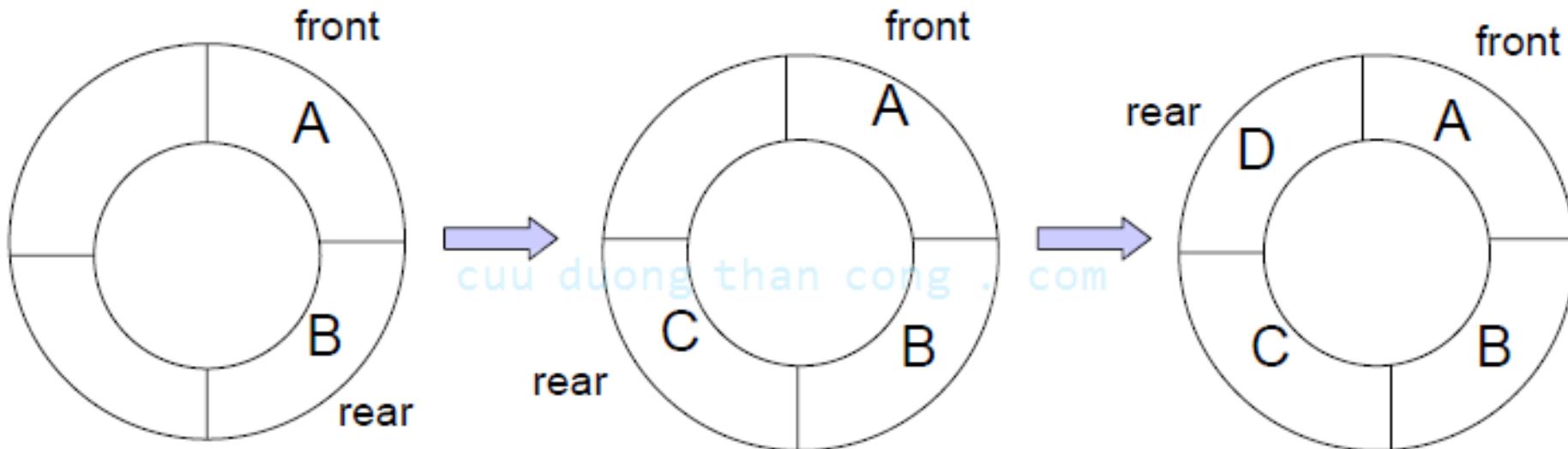


First array element

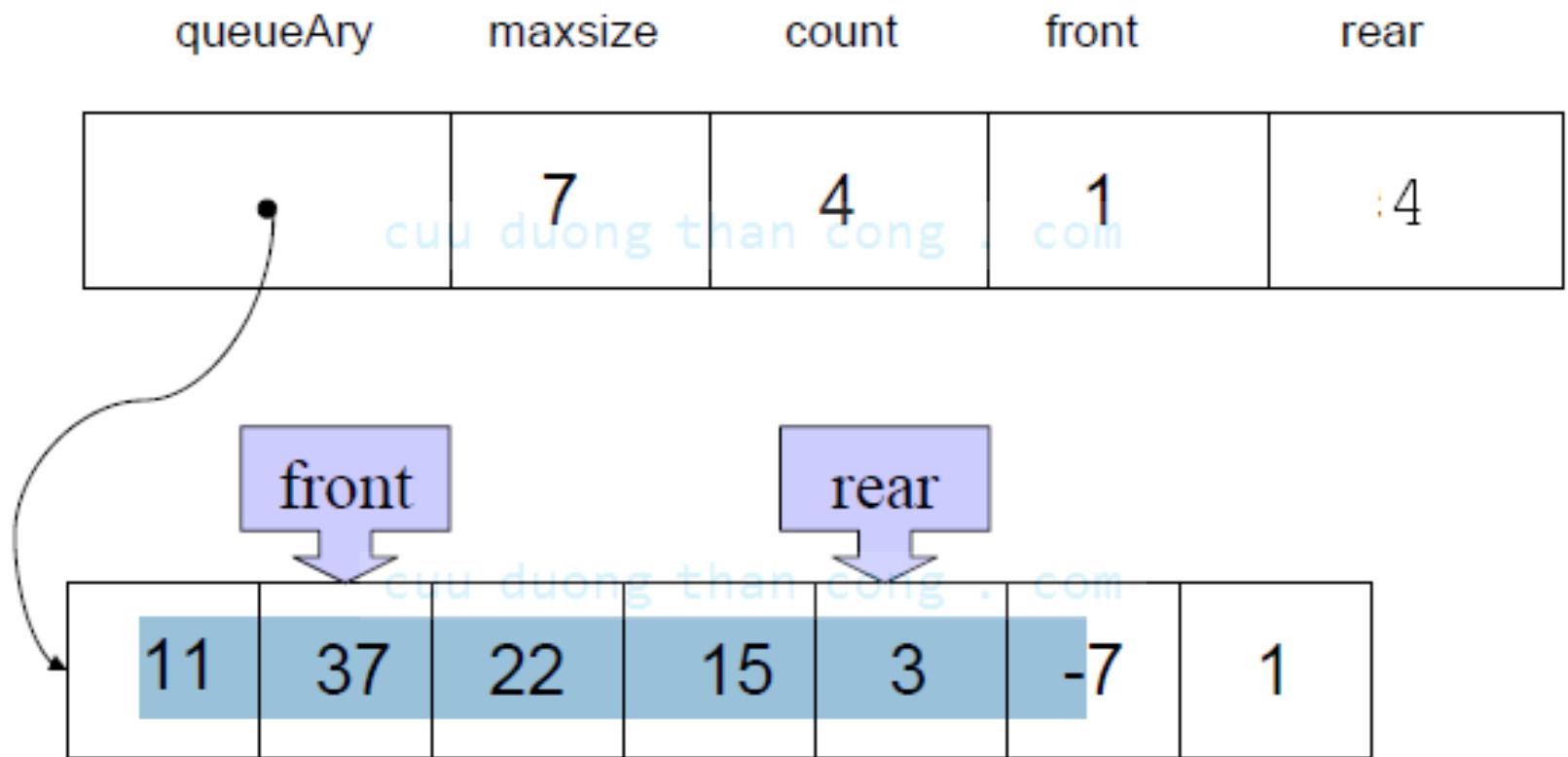
Last array element



Queue dạng vòng

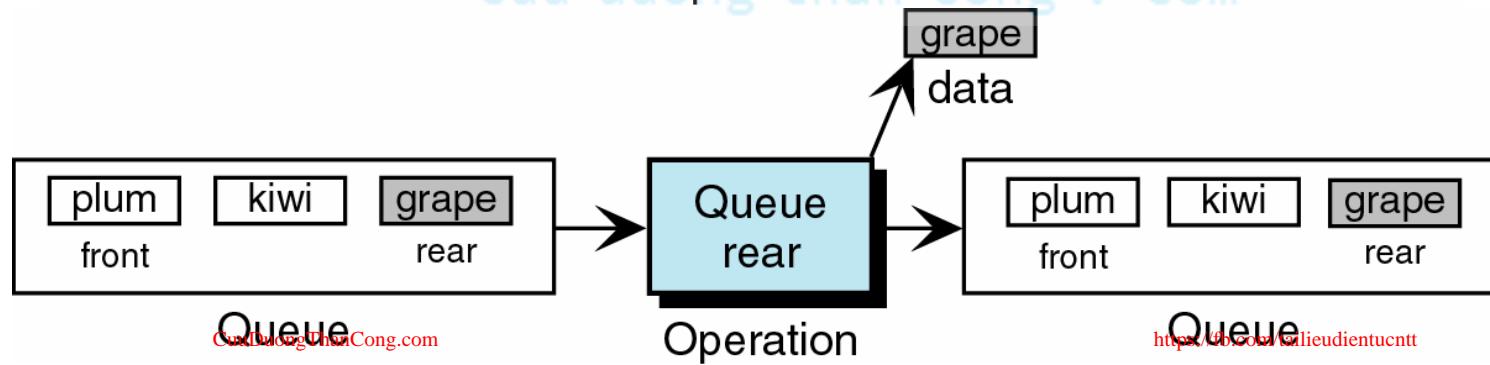
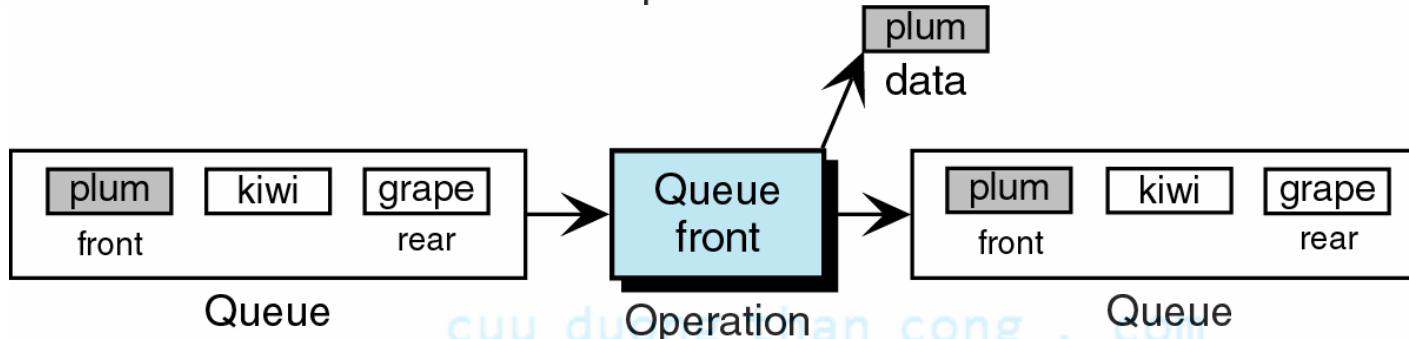
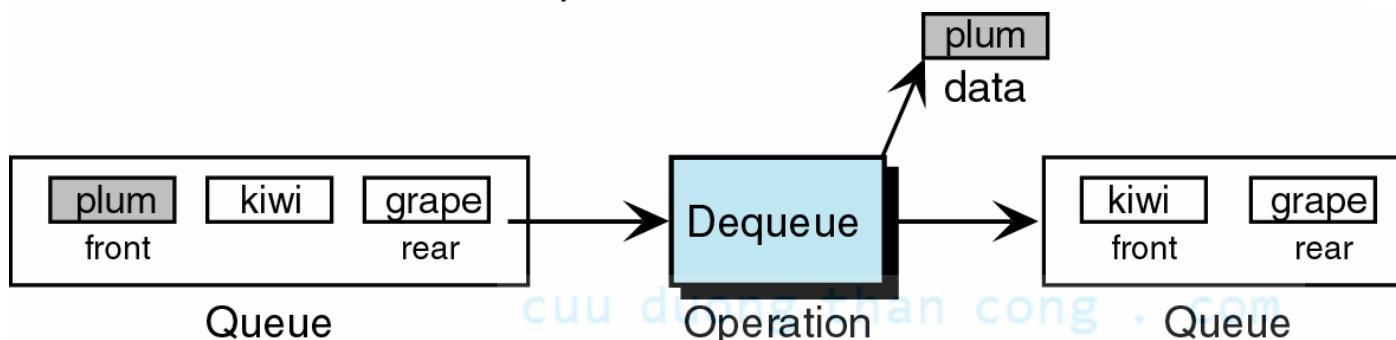
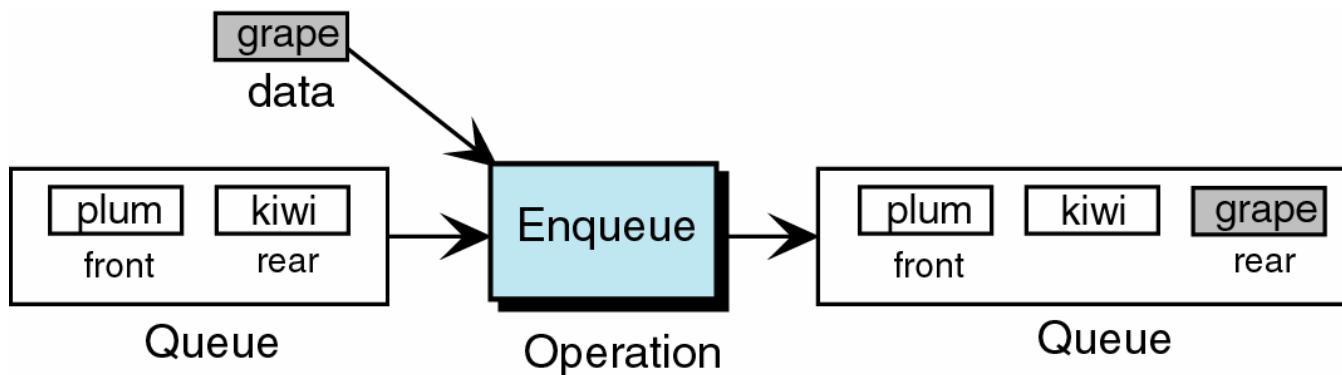


Queue thực hiện trên mảng



Các thao tác cơ bản với Queue

- Enqueue – Thêm một phần tử vào cuối queue
 - Tràn Overflow ?
- Dequeue – Xóa một phần tử tại đầu queue
 - Queue rỗng?
- Front – Trả lại phần tử tại đầu queue
 - Queue rỗng?
- Rear – Trả lại phần tử tại cuối queue
 - Queue rỗng



- Định nghĩa cấu trúc Queue

```
typedef struct intqueue{  
    Int *queueAry;  
    Int maxSize;      cuu duong than cong . com  
    Int count;  
    Int front;  
    Int rear;        cuu duong than cong . com  
} IntQueue;
```

- Tạo Queue

```
IntQueue *CreateQueue(int max){  
    IntQueue *queue;  
    queue = (IntQueue *)malloc(sizeof(IntQueue));  
    /*Cấp phát cho mảng */  
    queue->queueAry= malloc(max *sizeof(int));  
    /* Khởi tạo queue rỗng */  
    queue->front = -1;  
    queue->rear = -1;  
    queue->count = 0;  
    queue->maxSize= maxSize;  
    return queue;  
} /* createQueue*/
```

i queue

Int enqueue(struct intqueue *queue, int datain)

{

```
if (queue->count >= queue->maxSize) return 0;  
(queue->count)++;  
queue->rear = queue->rear % queue->maxSize + 1;  
queue->queueAry[queue->rear] = datain;  
return 1;
```

}

cuu duong than cong . com

Dequeue : Xóa PT ở đầu queue

```
int dequeue(struct intqueue *queue, int *dOutPtr)
{if(!queue->count)
    return 0;
*cuuduongthancong.com
*dOutPtr= queue->queueAry[queue->front];
(queue->count)--;
queue->front = (queue->front +1) % queue->maxSize;
return 1;
cuuduongthancong.com
}
```

- Front : Lấy pt đầu queue

```
Int Front(struct intqueue *queue,int *dOutPtr) {  
    if(!queue->count)  
        return 0;  
    else{  
        *dOutPtr= queue->queueAry[queue->front];  
        return 1;  
    }  
}
```

• Rear : lấy PT cuối Queue

```
int Rear(struct intqueue *queue,int*dOutPtr) {  
    if(!queue->count)  
        return 0;  
    else{  
        *dOutPtr= queue->queueAry[queue->rear];  
        return 1;  
    }  
}
```

- **emptyQueue và fullQueue**

```
Int emptyQueue(struct intqueue *queue)
```

```
{  
    return(queue->count == 0);  
}/* emptyQueue*/
```

```
Int fullQueue(struct intqueue *queue )
```

```
{  
    return( queue->count == queue->maxSize);  
}/* fullQueue*/
```

- **destroyQueue**

```
struct intqueue *destroyQueue(struct intqueue *queue)
{
    if(queue)
    {
        free(queue->queueAry);
        free(queue);
    }
    return NULL;
}/* destroyQueue*/
```

Bài tập

- Xây dựng Stack và Queue mốc nối, cài đặt các thao tác tương ứng

cuu duong than cong . com

cuu duong than cong . com

4.II.2 Tree

1. Định nghĩa và khái niệm

2. Cây nhị phân

- Định nghĩa và Tính chất
- Lưu trữ
- Duyệt cây

3. Cây tổng quát

- Biểu diễn cây tổng quát
- Duyệt cây tổng quát (nói qua)

4. Ứng dụng của cấu trúc cây

- Cây biểu diễn biểu thức (tính giá trị, tính đạo hàm)
- Cây quyết định

1. Định nghĩa và khái niệm

- So với cấu trúc liên tục như mảng, danh sách có ưu điểm vượt trội về tính mềm dẻo
- Nhưng nhược điểm lớn của ds là tính tuần tự và chỉ thể hiện được các mối quan hệ tuyến tính.
- Thông tin còn có thể có quan hệ dạng phi tuyến, ví dụ:
 - Các thư mục file
 - Các bước di chuyển của các quân cờ
 - Sơ đồ nhân sự của tổ chức
 - Cây phả hệ
- Sử dụng cây cho phép tìm kiếm thông tin nhanh

Các khái niệm cơ bản về cây

- Một **cây** (*tree*) gồm một tập hữu hạn các **nút** (*node*) và 1 tập hữu hạn các **cành** (*branch*) p “cha-con”.
- Số cạnh ra (con) tại một nút gọi là **p** (*degree*) của nút đó. Nếu cây không rỗng thì phải có 1 nút gọi là **nút gốc** (*root*), nút này **không có cạnh vào**
- t trên
- cây.
- **Định nghĩa (ĐQ):** Một cây là tập các nút mà :
 - là tập rỗng, hoặc
 - có 0 hoặc nhiều cây con, các cây con cũng là cây.

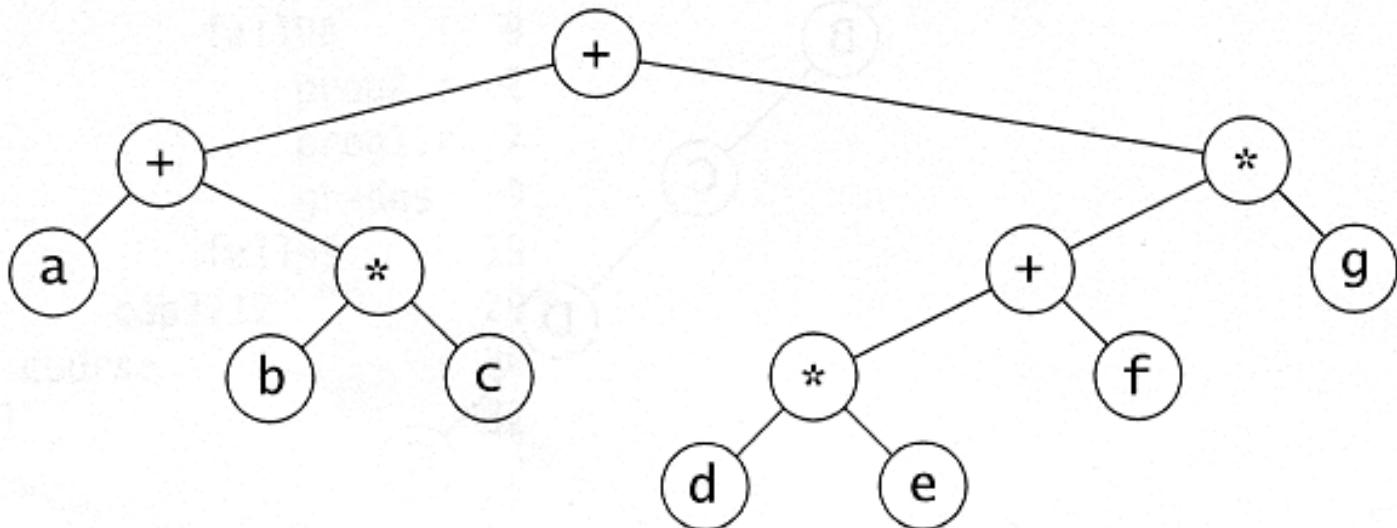


Figure 4.14 Expression tree for $(a + b * c) + ((d * e + f) * g)$

cuu duong than cong . com

M (A (N C (B)) D O (Y (T X) E L S))
(c)

M

- A

- - N

- - C

- - - B

- D

- O

- - Y

- - - T

- - - X

- - E

- - L

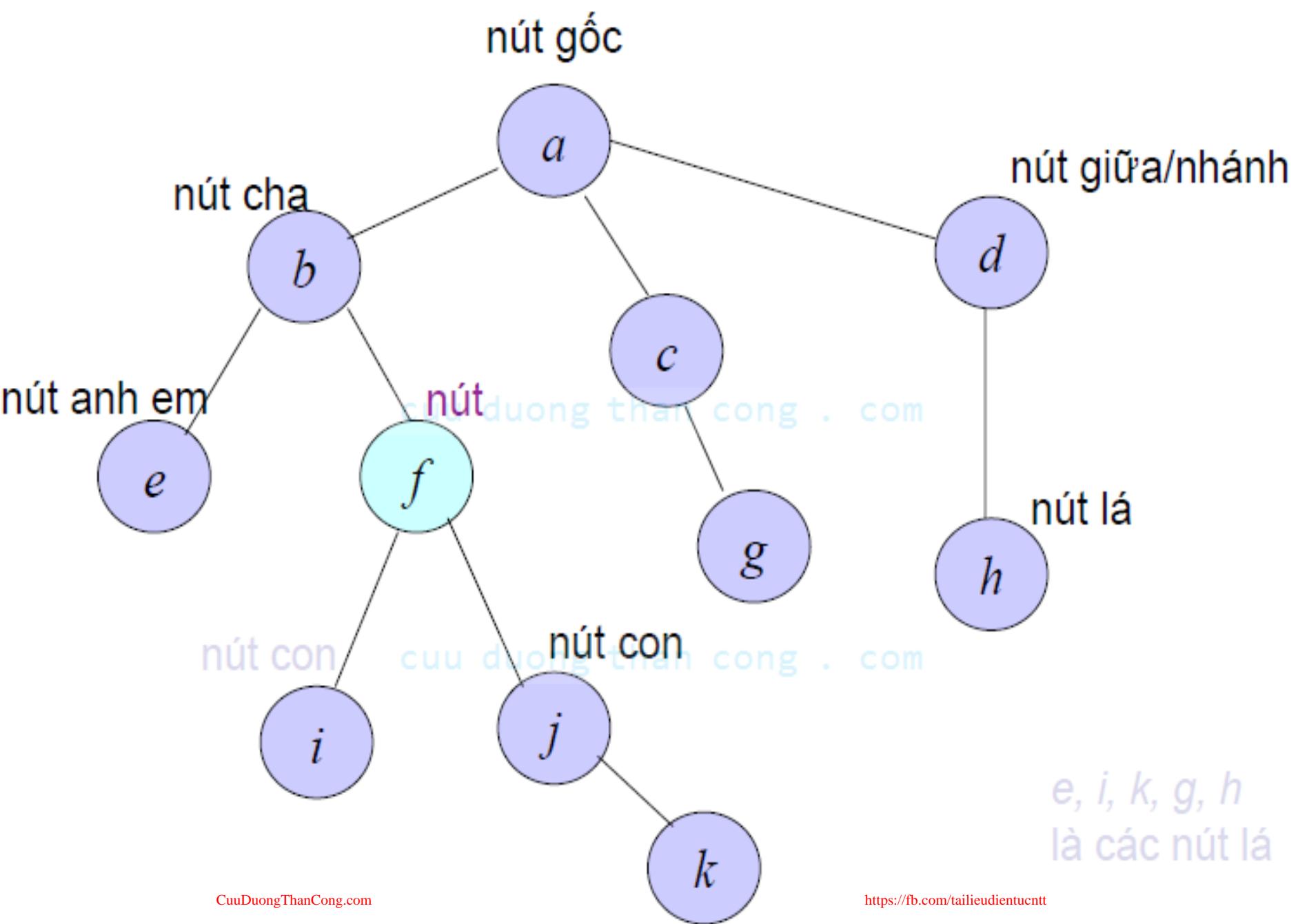
- - S

(b)

05/04/2019

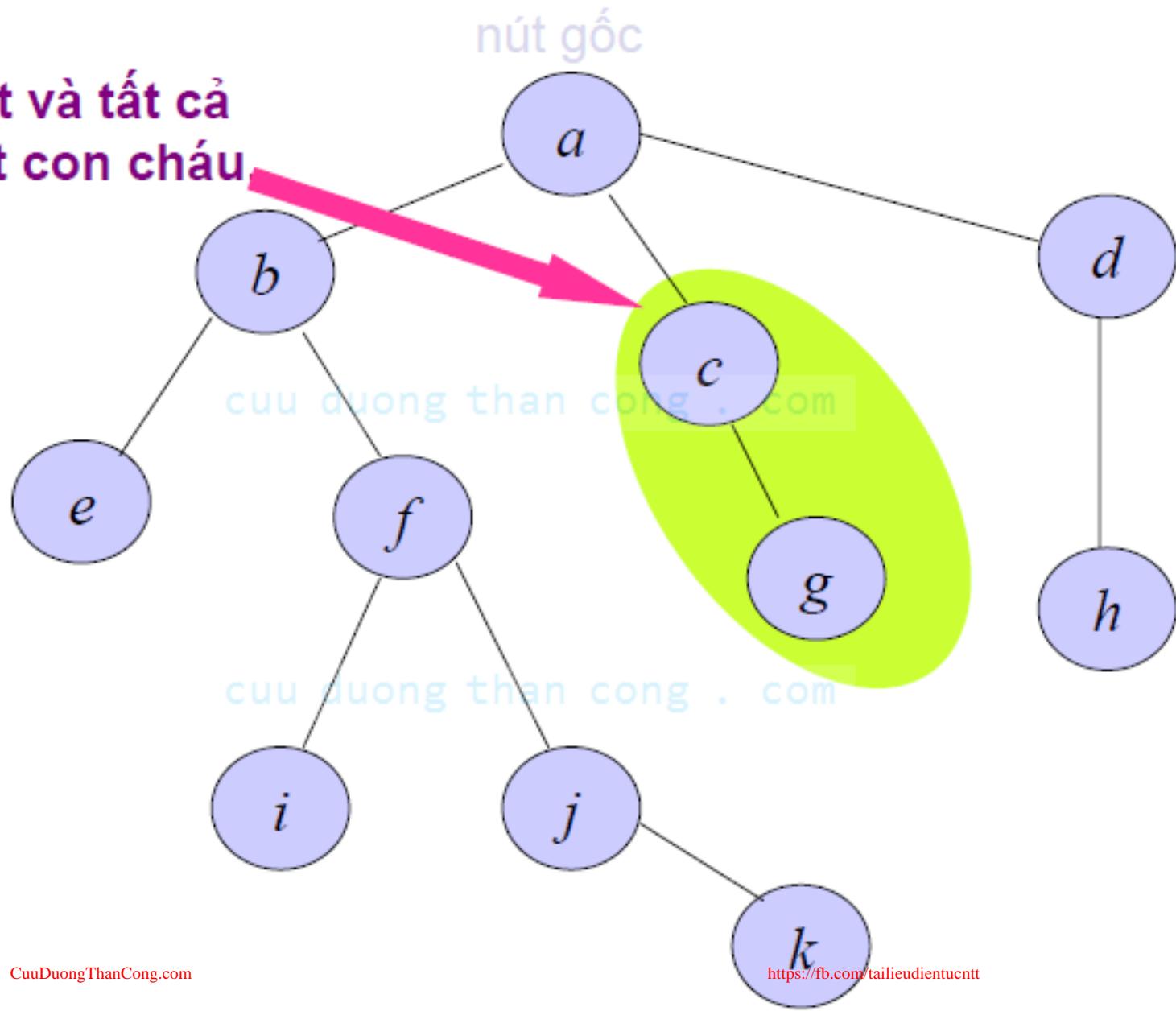
Last Update 8-2010
CuuDuongThanCong.com

Các cách biểu diễn cây



Cây con

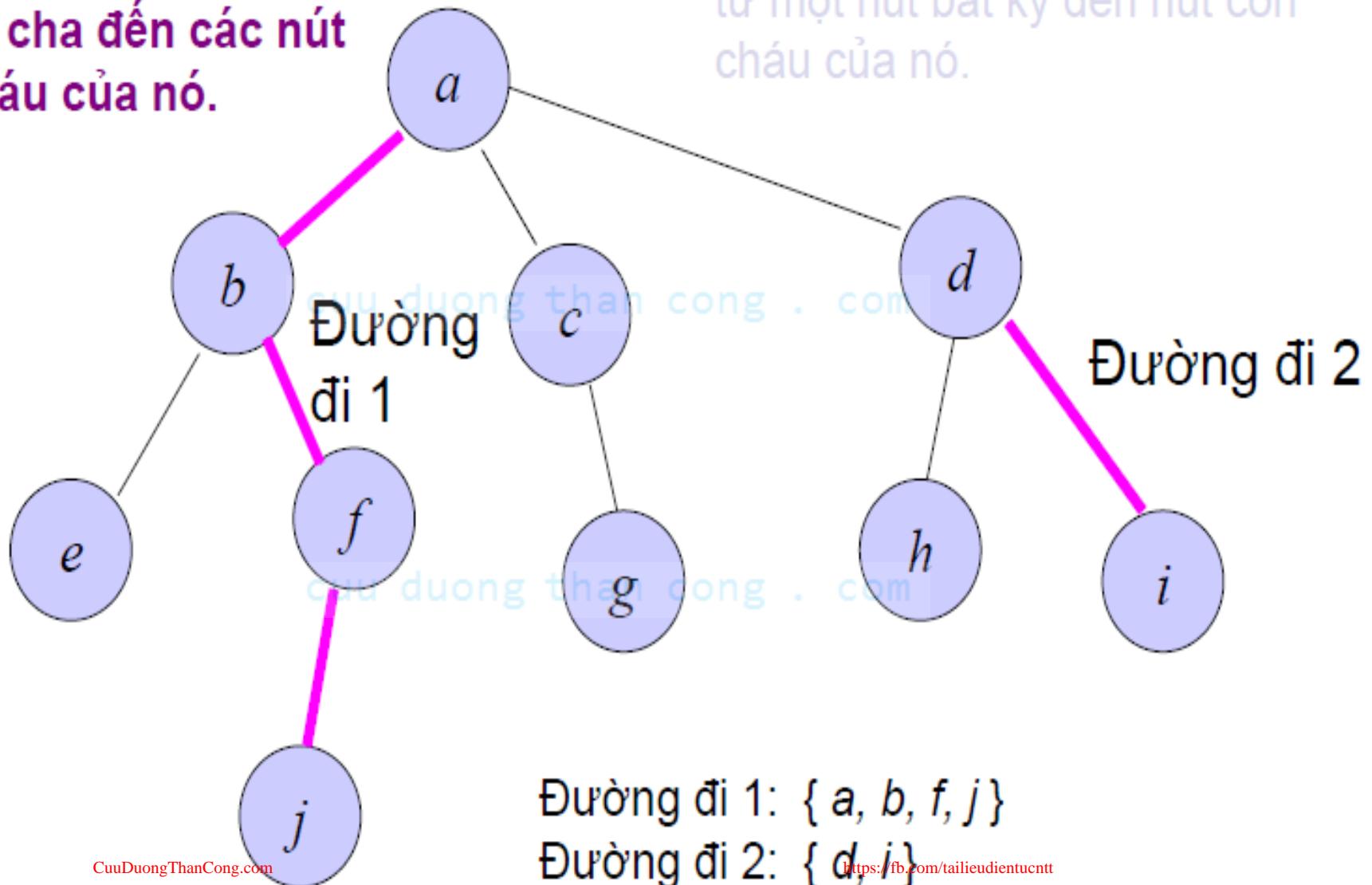
Một nút và tất cả các nút con cháu



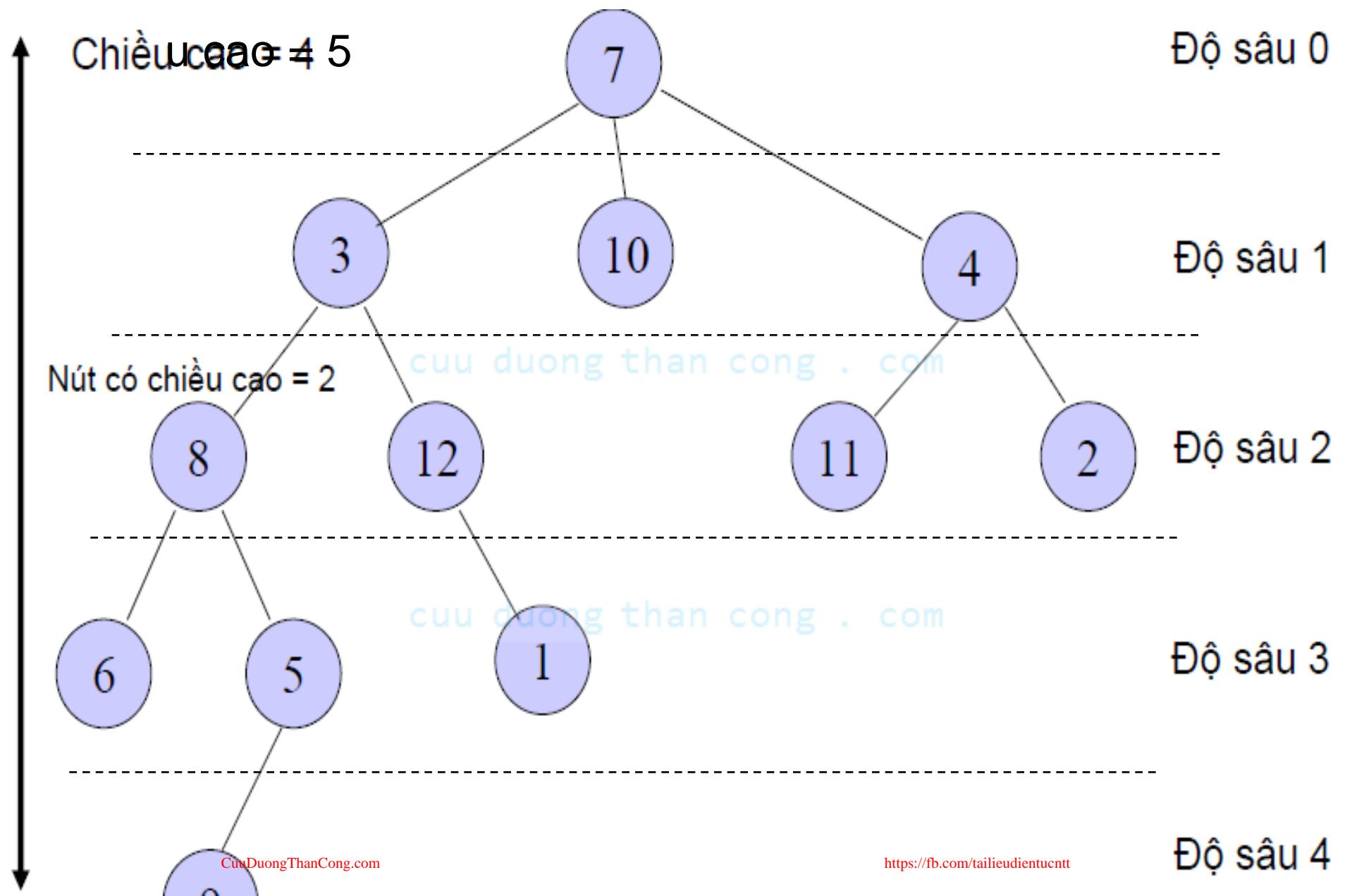
Đường đi

Từ nút cha đến các nút con cháu của nó.

Tồn tại một **đường đi duy nhất** từ một nút bất kỳ đến nút con cháu của nó.

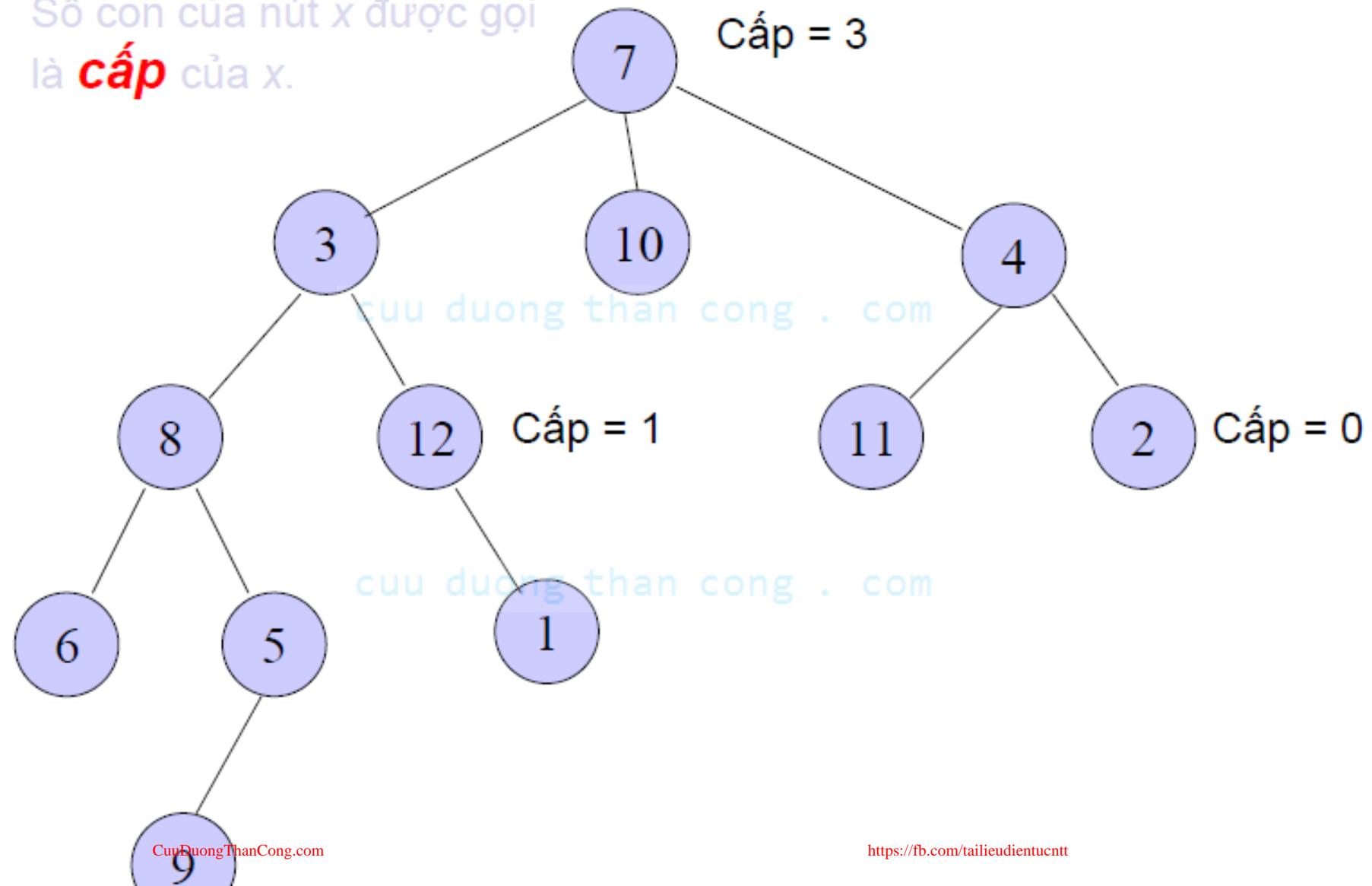


Độ sâu và chiều cao



Cấp

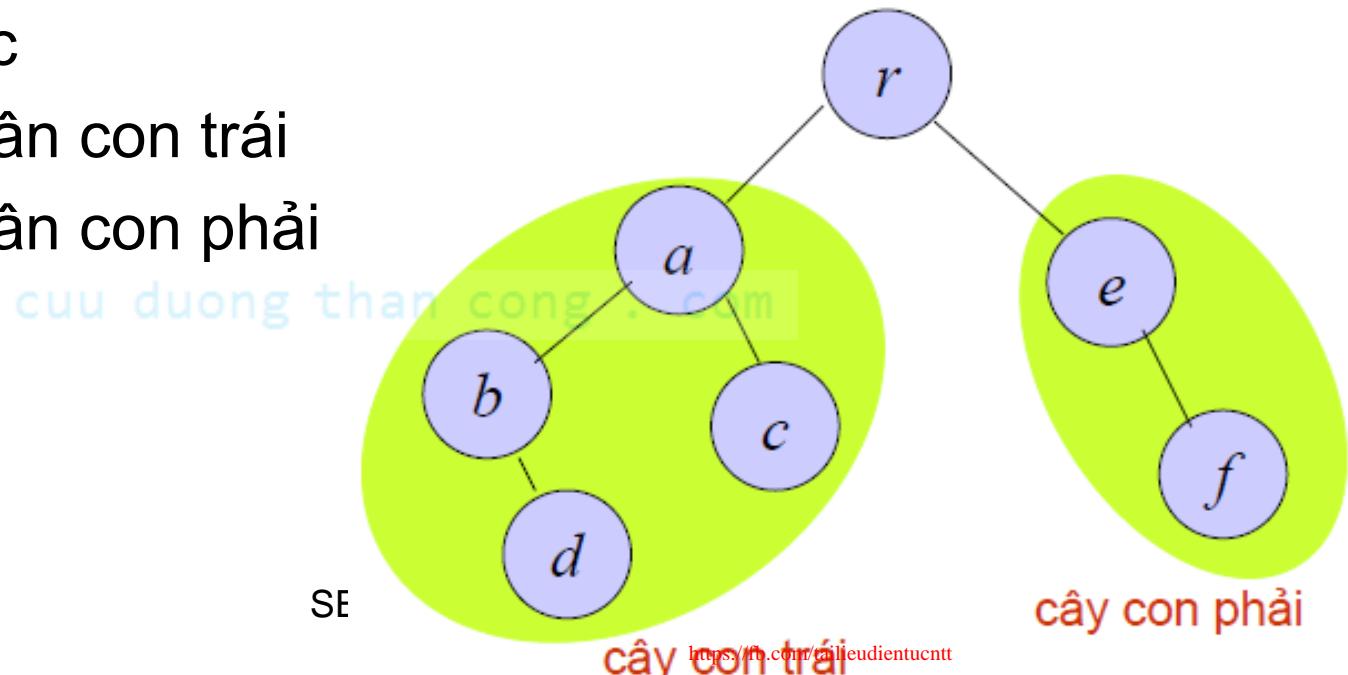
Số con của nút x được gọi
là **cấp** của x.



2. Cây nhị phân

2.1. Định nghĩa và tính chất

- Mỗi nút có nhiều nhất 2 nút con. Nút trái và nút phải
- Một tập các nút T được gọi là cây nhị phân, nếu :
 - a) Nó là cây rỗng, hoặc
 - b) Gồm 3 tập con không trùng nhau:
 - 1) một nút gốc
 - 2) Cây nhị phân con trái
 - 3) Cây nhị phân con phải

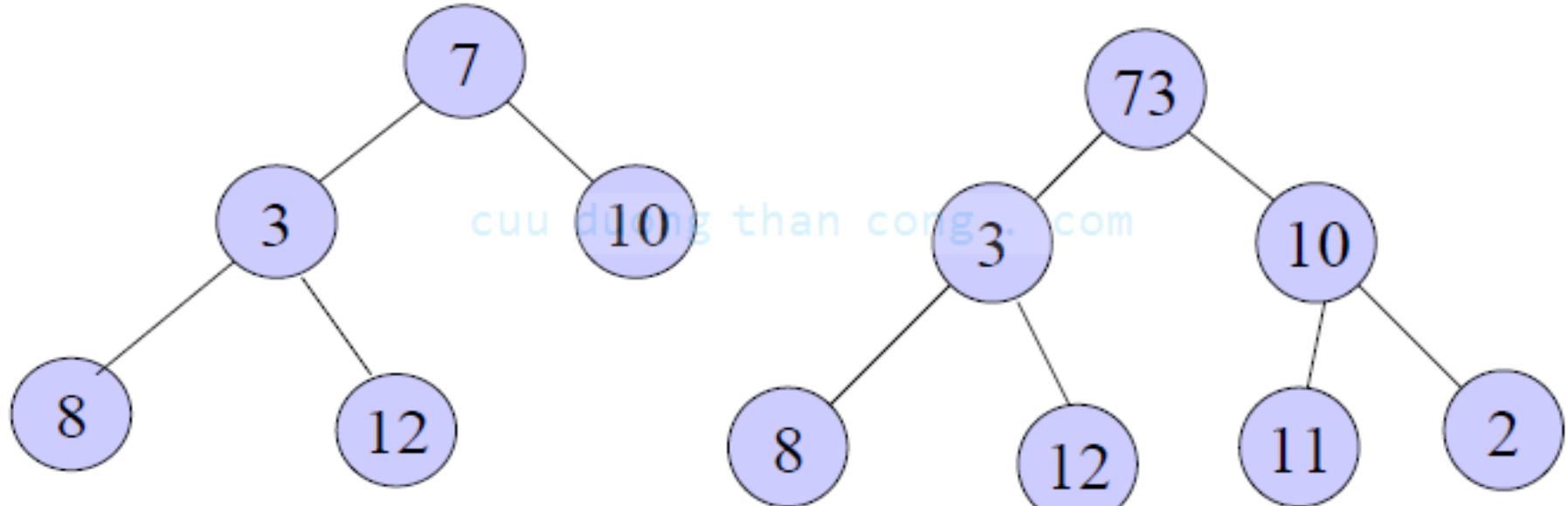


Cây nhị phân hoàn chỉnh và Cây nhị phân đầy đủ

i đa 2 con

Tất cả nút lá đều có cùng
độ sâu và tất cả nút
giữa có cấp = 2

cuu duong than cong . com



Một số tính chất

- $c_i : 2^{i-1}$
-

Độ cao H thì $N = 2^H - 1$
– $H_{\max} = N$, $H_{\min} = \lceil \log_2 N \rceil + 1$

phân

c)

- Khoảng cách từ 1 nút đến nút gốc xác định chi phí cần để định vị nó : 1 nút có độ sâu là 5 => phải đi từ nút gốc và qua 5 cành
- cuu duong than cong . com

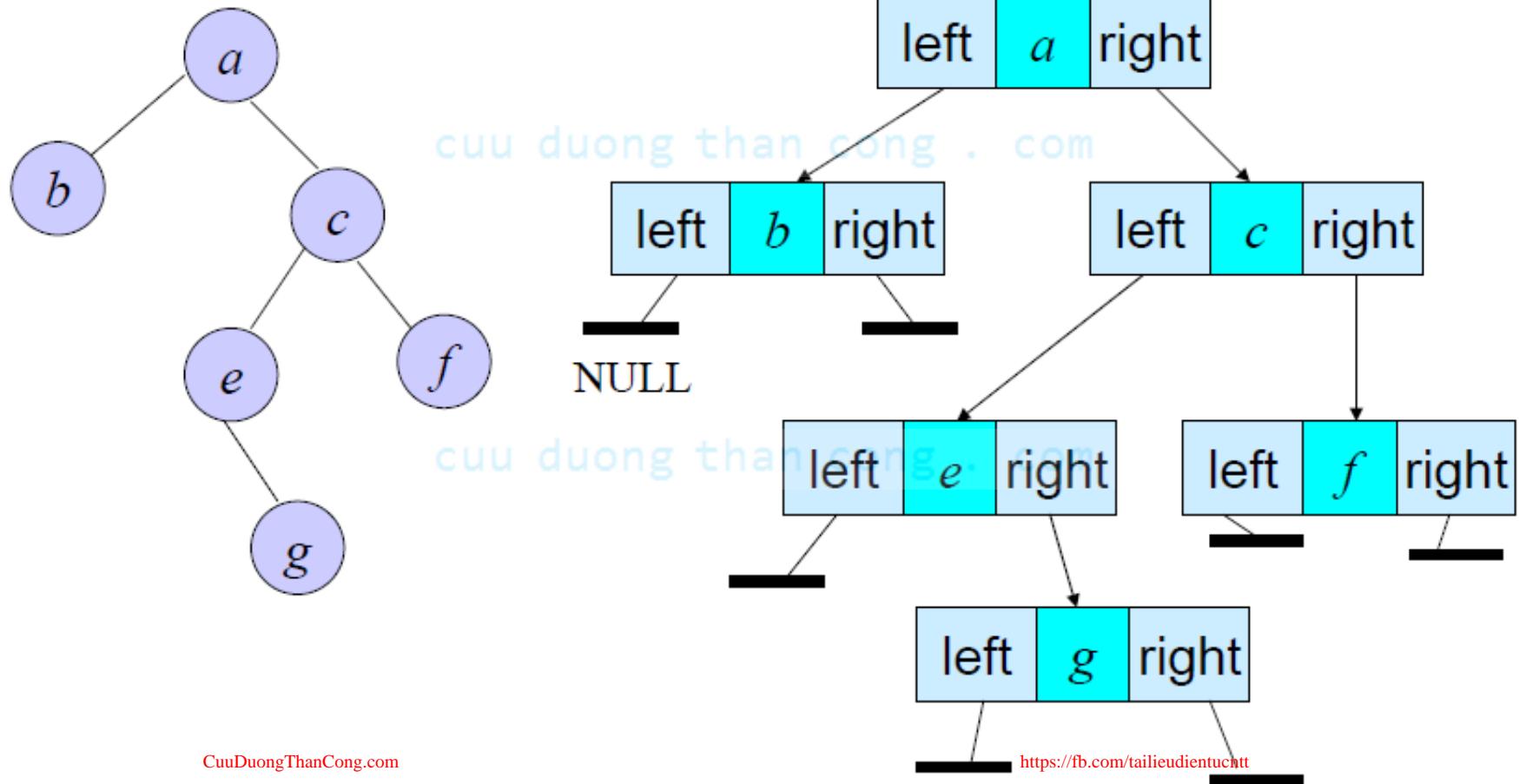
ng của cây nhị phân. Hệ số cân bằng của cây (*balance factor*) chênh lệch giữa chiều cao của 2 cây con trái và phải của nó:

$$B = HL - HR$$

- Một cây cân bằng khi $B = 0$ và các cây con của nó cũng cân bằng

2.2 Lưu trữ cây nhị phân

- Lưu trữ kế tiếp : Sử dụng mảng
- Lưu trữ mốc nối : Sử dụng con trỏ

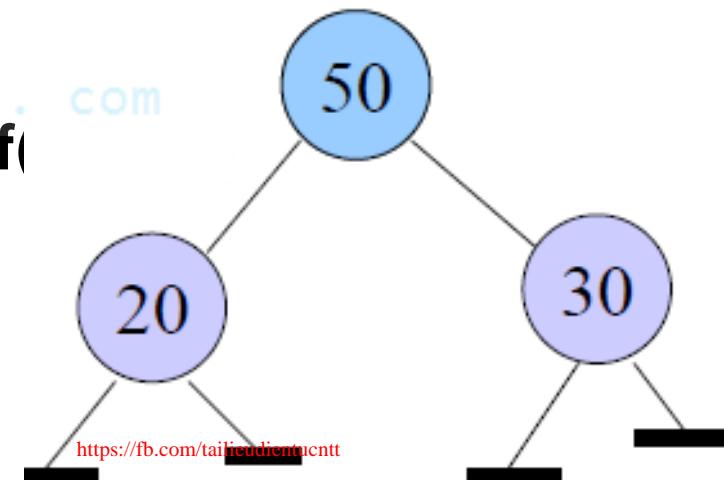


- Cấu trúc cây nhị phân

```
typedef structtree_node
{
    int data ;
    structtree_node *left ;
    structtree_node *right ;
}TREE_NODE;
```

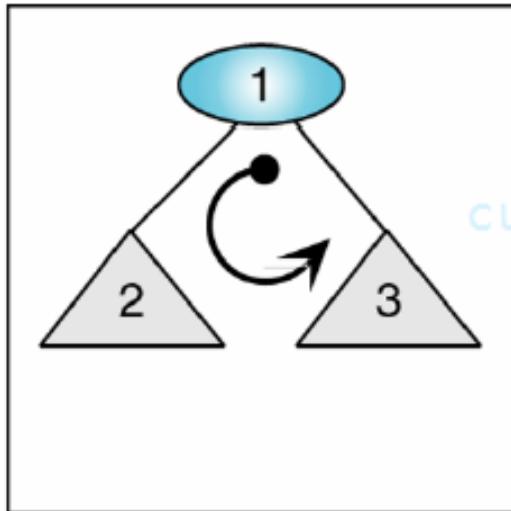
• Tạo cây nhị phân

```
TREE_NODE *root, *leftChild, *rightChild;  
// Tạo nút con trái  
leftChild= (TREE_NODE *)malloc(sizeof(TREE_NODE));  
leftChild->data = 20;  
leftChild->left = leftChild->right = NULL;  
// Tạo nút con phải  
rightChild = (TREE_NODE *)malloc(sizeof(TREE_NODE));  
rightChild->data = 30;  
rightChild->left = rightChild->right = NULL;  
// Tạo nút gốc  
root = (TREE_NODE *)malloc(sizeof(  
root->left = leftChild;  
root->right = rightChild;  
root-> data= 50;// gán 50 cho root
```

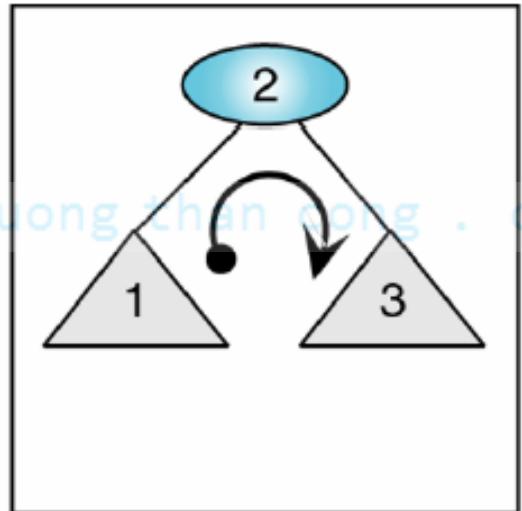


2.3. Duyệt cây nhị phân

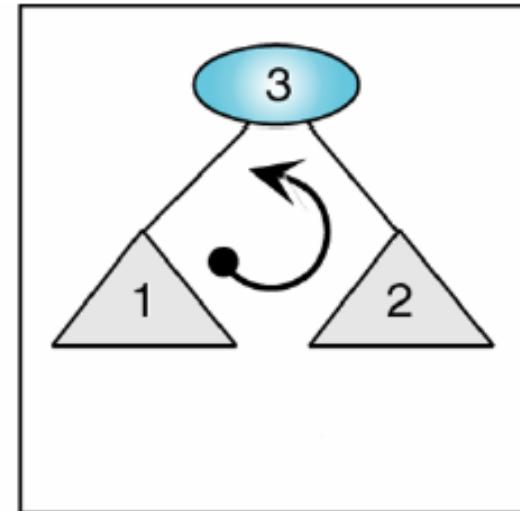
- Duyệt cây: lần lượt duyệt toàn bộ nút trên cây
- Có 3 cách duyệt cây:
 - Duyệt theo thứ tự trước
 - Duyệt theo thứ tự giữa
 - Duyệt theo thứ tự sau
- Định nghĩa duyệt cây nhị phân là những định nghĩa đệ quy.



(a) Thứ tự trước



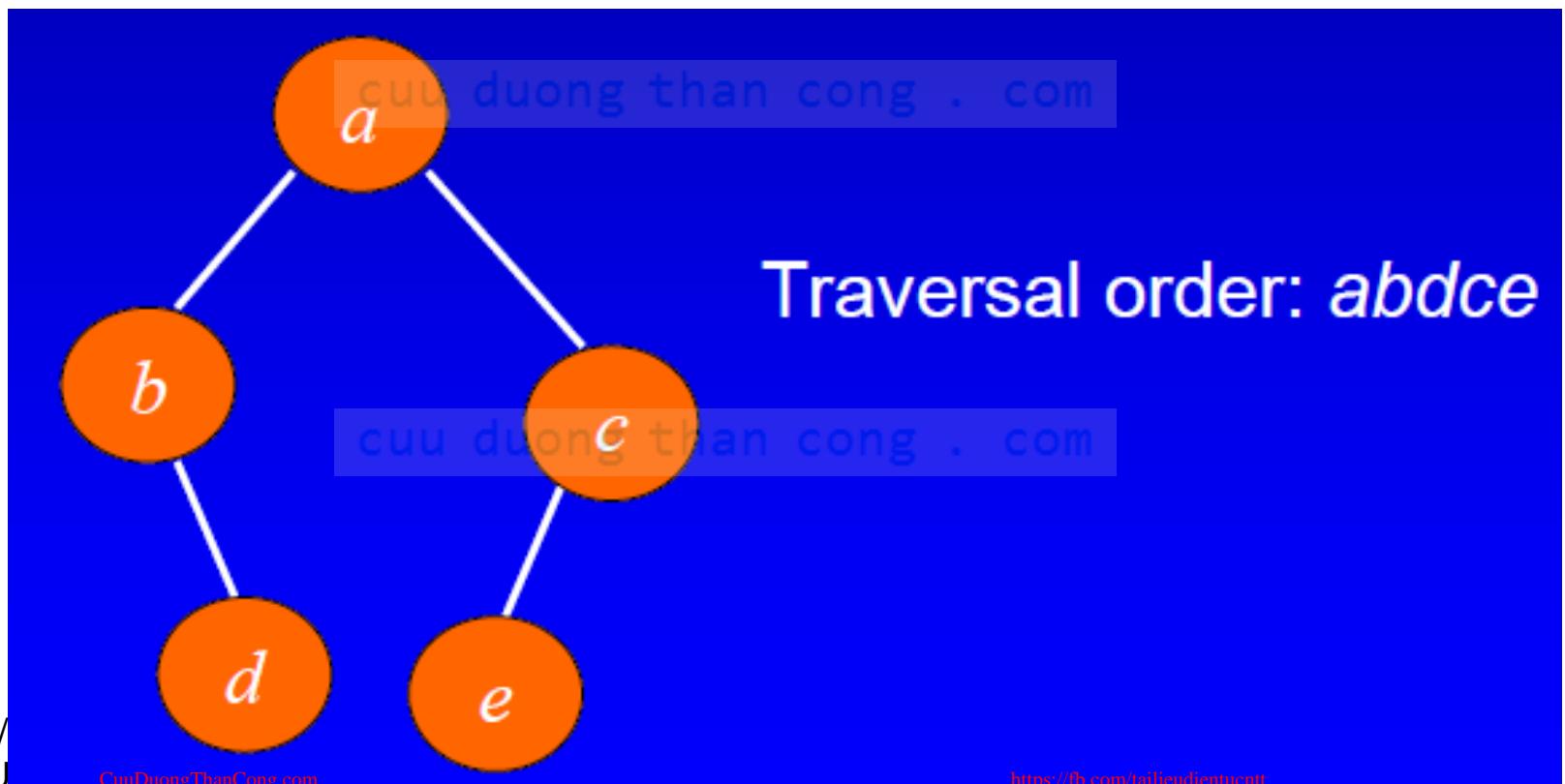
(b) Thứ tự giữa



(c) Thứ tự sau

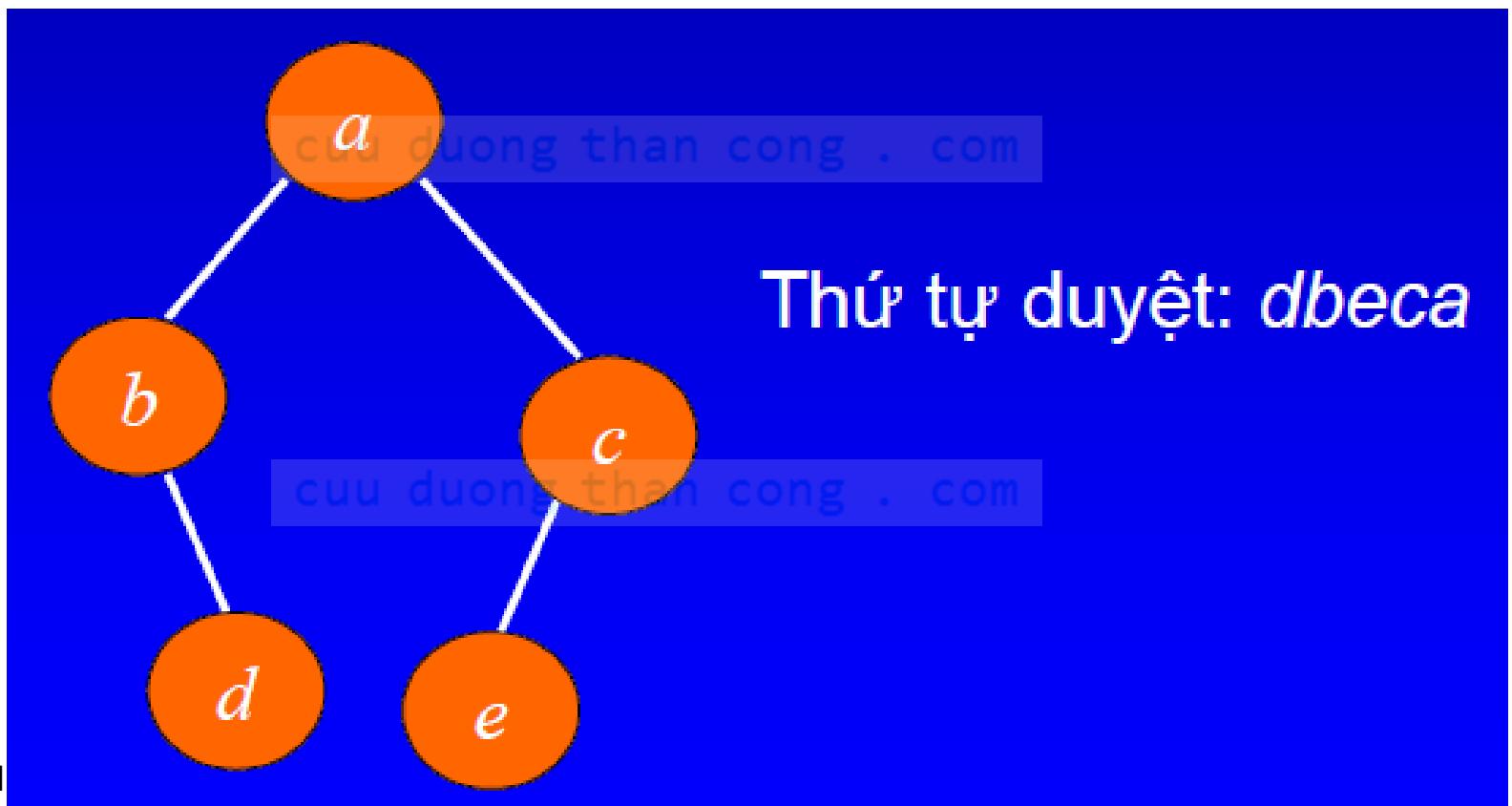
Duyệt theo thứ tự trước

1. Thăm nút.
2. Duyệt cây con trái theo thứ tự trước.
3. Duyệt cây con phải theo thứ tự trước.



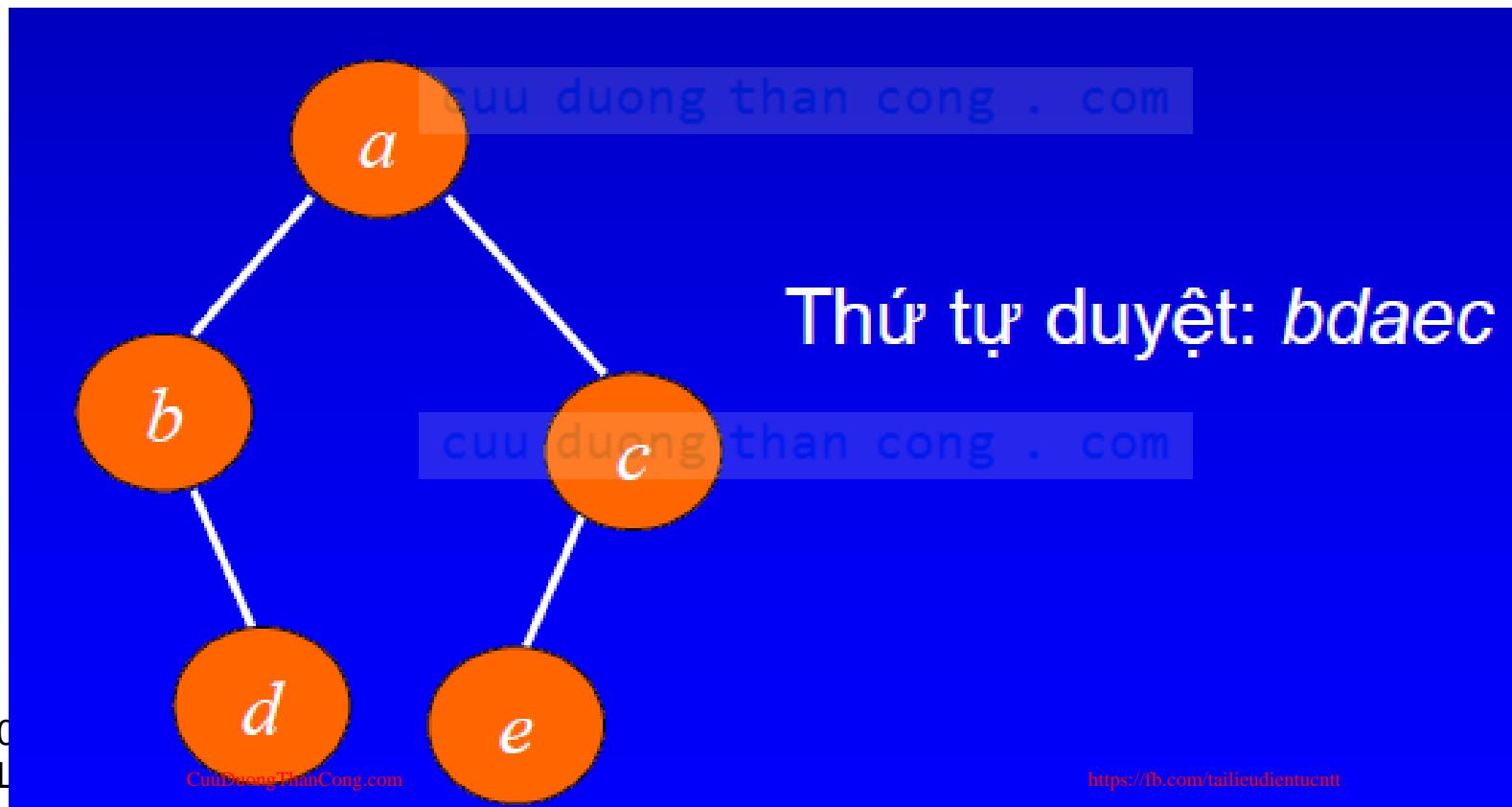
Duyệt theo thứ tự sau

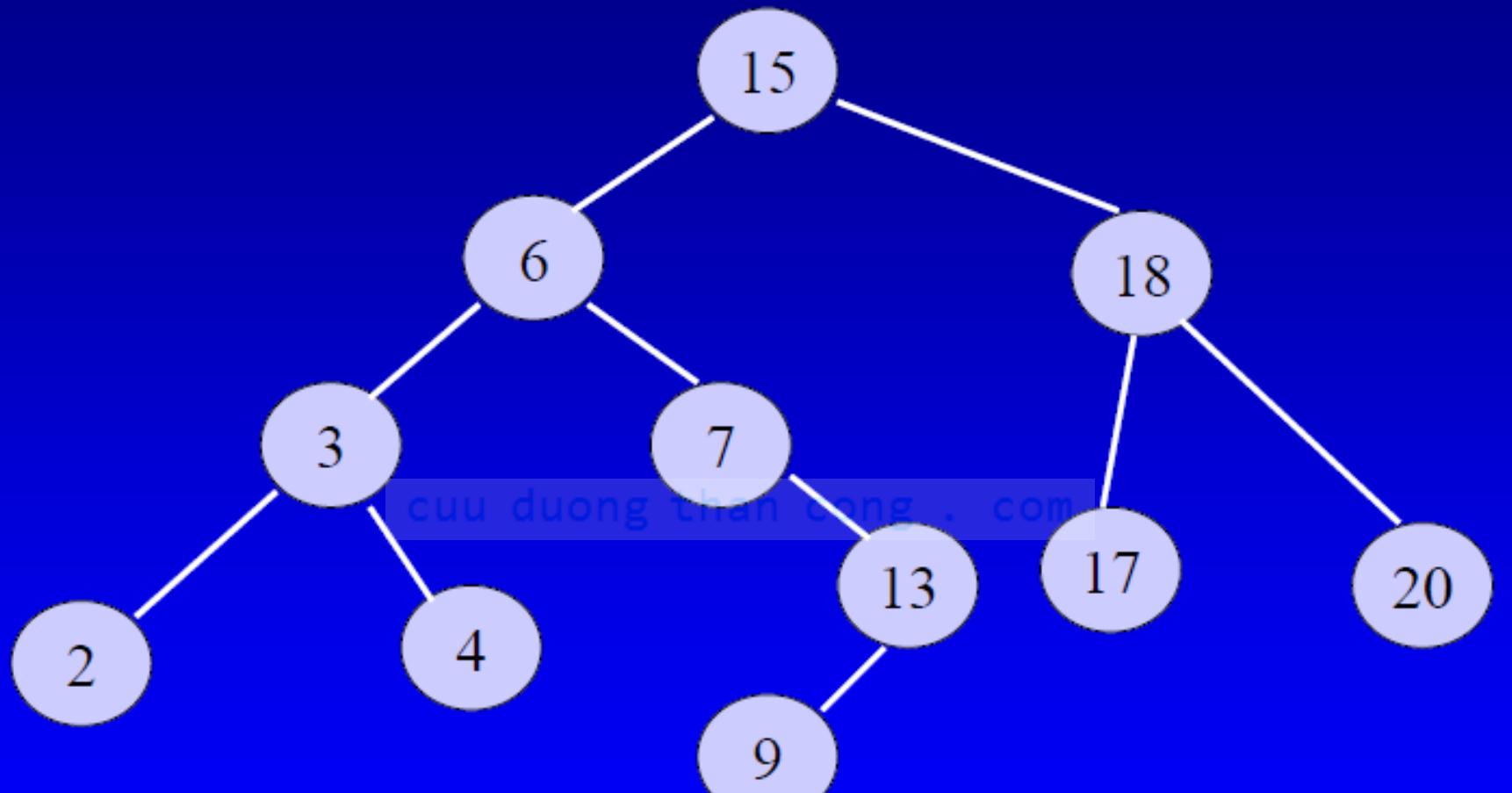
1. Duyệt cây con trái theo thứ tự sau.
2. Duyệt cây con phải theo thứ tự sau.
3. Thăm nút.



Duyệt theo thứ tự giữa

1. Duyệt cây con trái theo thứ tự giữa
2. Thăm nút.
3. Duyệt cây con phải theo thứ tự giữa.





Thứ tự trước: 15, 6, 3, 2, 4, 7, 13, 9, 18, 17, 20

Thứ tự giữa :2, 3, 4, 6, 7, 9, 13, 15, 17, 18, 20

Thứ tự sau :2, 4, 3, 9, 13, 7, 6, 17, 20, 18, 15

Duyệt theo thứ tự trước—Đệ quy

```
void Preorder(TREE_NODE *root)
{
    if (root != NULL)
    {
        // tham aNode
        printf("%d", root->data);
        // duyet cay con trai
        Preorder(root->left);
        // duyet cay con phai
        Preorder(root->right);
    }
}
```

Bài tập: Viết giải thuật đệ quy của

- ☐ Duyệt theo thứ tự giữa
- ☐ Duyệt theo thứ tự sau

Duyệt theo thứ tự trước—Vòng lặp

```
void Preorder_iter(TREE_NODE *treeRoot)
```

i ưu!!! Xem trong CSDL>

```
    TREE_NODE *curr= treeRoot;
    STACK *stack= createStack(MAX);// khởi tạo stack
    while (curr != NULL || !IsEmpty(stack))
    {
        printf("%d", curr->data); // thăm curr
        // nếu có cây con phải, đẩy cây con phải vào stack
        if (curr->right != NULL)
            pushStack(stack, curr->right);
        if(curr->left!=NULL)
            curr= curr->left; // duyệt cây con trái
        else
            popStack(stack, &curr); // duyệt cây con phải
    }
    destroyStack(&stack); // giải phóng stack
```

• Duyệt theo thứ tự giữa

```
void Inorder_iter(TREE_NODE *root){  
    TREE_NODE *curr= root;  
    STACK *stack= createStack(MAX);//ktạo stack  
    while(curr != NULL || !IsEmpty(stack))  
    {  
        if (curr==NULL){  
            popStack(stack, &curr);  
            printf("%d", curr->data);  
            curr= curr->right;  
        }  
        else{  
            pushStack(stack, curr);  
            curr= curr->left; // duyệt cây con trái  
        }  
    }  
    destroyStack(stack); //giải phóng stack
```

Duyệt theo thứ tự cuối

```
voidPostorder_iter(TREE_NODE *treeRoot)
{
    TREE_NODE *curr= treeRoot;
    STACK *stack= createStack(MAX);//ktạo một stack
    while( curr != NULL || !IsEmpty(stack)) {
        if (curr == NULL) {
            while(!IsEmpty(stack) && curr==Top(stack)->right){
                cuu = PopStack(stack, &curr);
                printf("%d", curr->data);
            }
            curr= isEmpty(stack)? NULL: Top(stack)->right;
        }
        else{
            PushStack(stack, curr);
            curr= curr->left;
        }
    }
    destroyStack(&stack);// giải phóng stack
}
```

05/04/2019
Last Update 8-2010
CodeDongThanCong.com

SE-SolCT

KLT4-2.102
<https://fb.com/tailieudientucntt>

Một vài ứng dụng của duyệt cây

- 1.Tính độ cao của cây
- 2.Đếm số nút lá trong cây
- 3.Sao chép cây [cuuduongthancong . com](http://cuuduongthancong.com)
- 4.Xóa cây

cuuduongthancong . com

Tính độ cao của cây

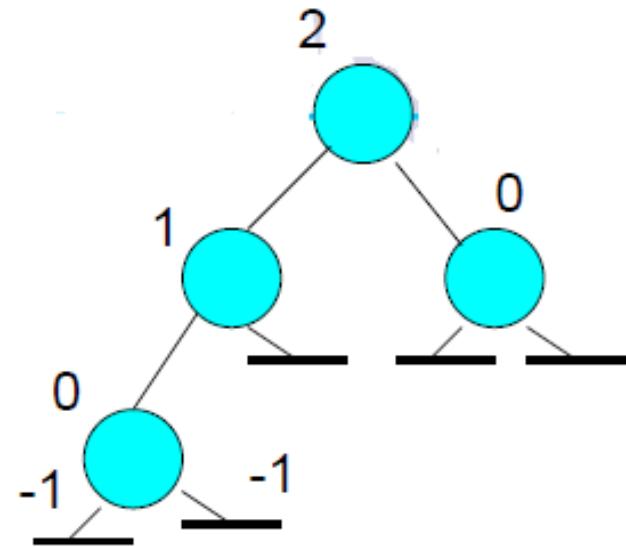
```
int Height(TREE_NODE *tree)
{
    Int heightLeft, heightRight, heightval;
    if( tree== NULL )
        heightval= 0;
    else
    { // Sử dụng phương pháp duyệt theo thứ tự sau
        heightLeft= Height (tree->left);
        heightRight= Height (tree->right);
        heightval= 1 + max(heightLeft,heightRight);
    }
    return heightval;
}
```

05/04/2019

Last Update 8-2010
CuuDuongThanCong.com

SE-SolICT

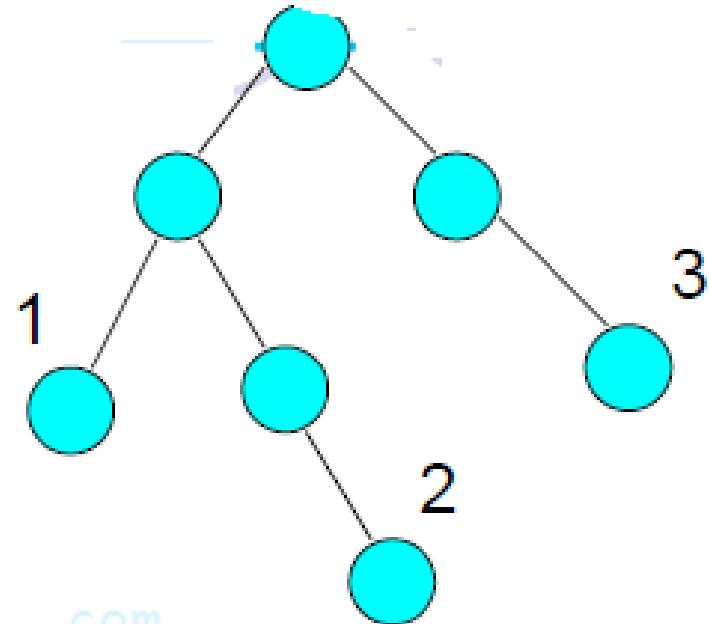
<https://fb.com/tailieudientucntt>



KLT4-2.104

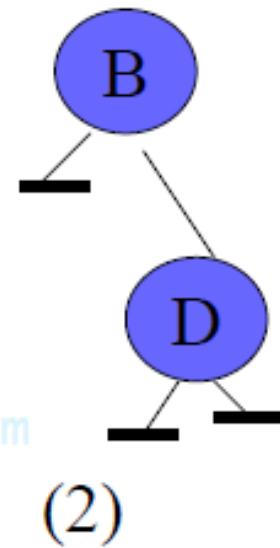
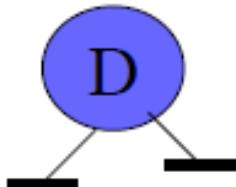
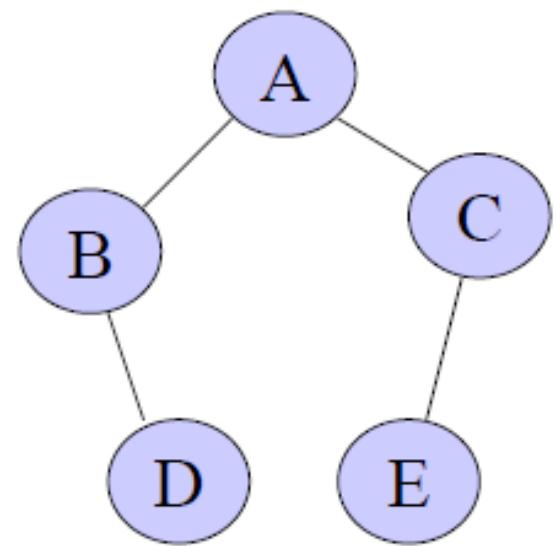
Đếm số nút

```
int Count(TREE_NODE *tree)
{
    if (tree == NULL)    return 0;
    else {
        int count;
        count = 1 + Count(tree->left) + Count(tree->right);
        return count;
    }
}
```

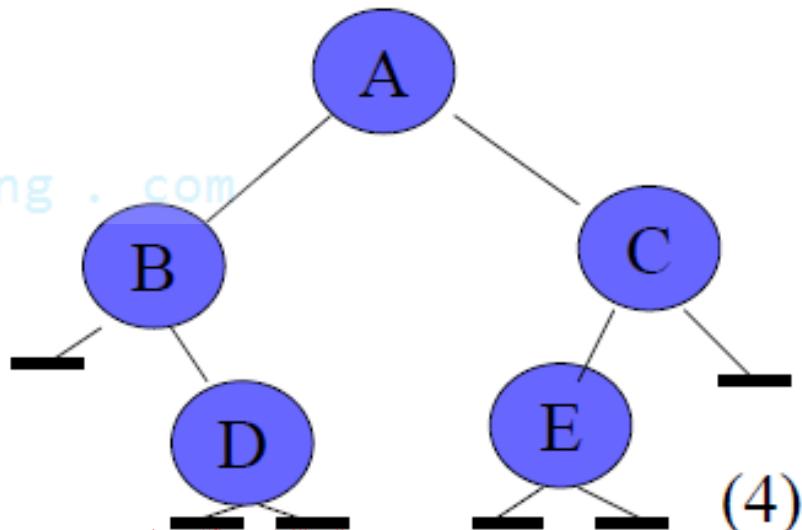
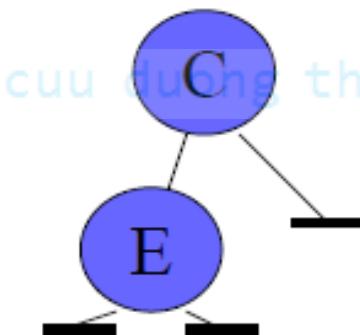
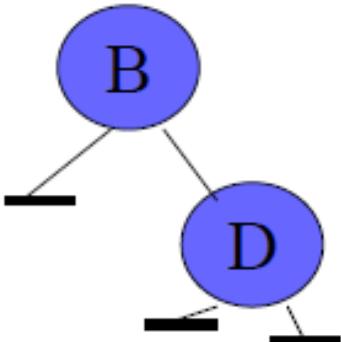


a cây!!!

Sao chép cây



cuu duong than cong . com
(1)



```
TREE_NODE *CopyTree(TREE_NODE *tree)
{
    // Dùng đệ quy khi cây rỗng
    if (tree== NULL) return NULL;
    TREE_NODE *leftsub, *rightsub, *newnode;
    leftsub=CopyTree(tree->left);
    rightsub= CopyTree(tree->right);
    // tạo cây mới
    newnode= malloc(sizeof(TREE_NODE));
    newnode->data = tree->data;
    newnode->left = leftsub;
    newnode->right = rightsub;
    return newnode;
}
```

Xóa cây

```
void DeleteTree(TREE_NODE *tree)
{
    //xóa theo thứ tự sau
    if(tree != NULL) cuu duong than cong . com
    {
        DeleteTree(tree-> left);
        DeleteTree(tree-> right);
        free(tree); cuu duong than cong . com
    }
}
```

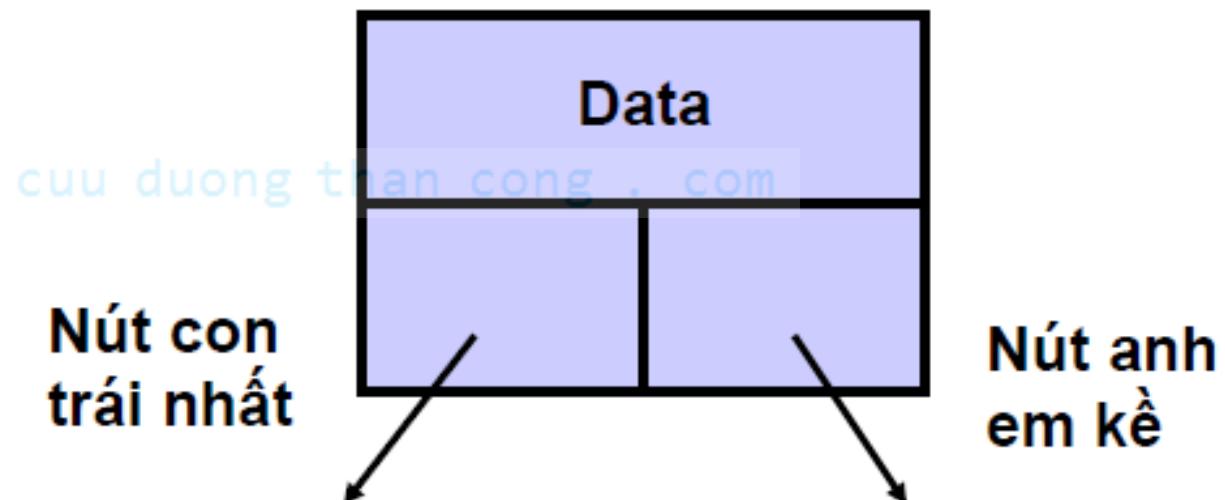
3. Cây tổng quát

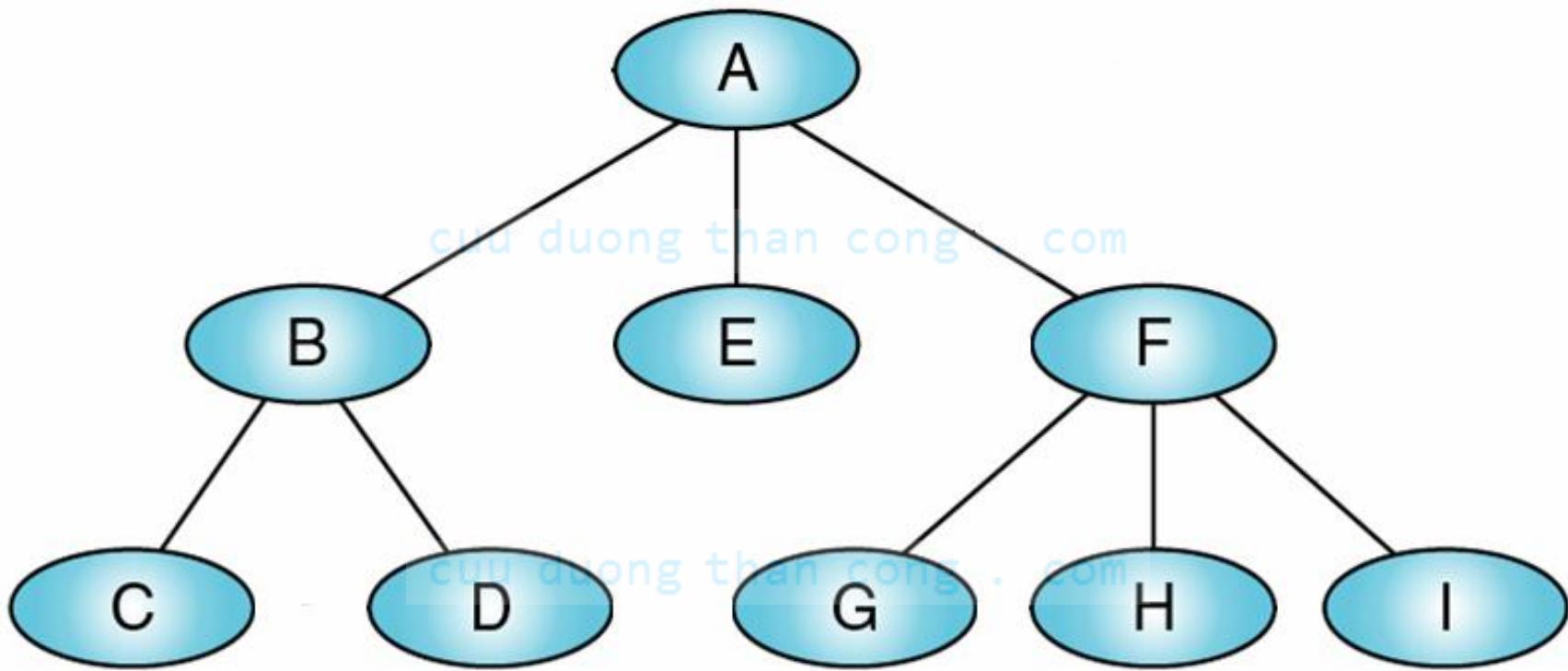
3.1. Biểu diễn cây tổng quát

- Biểu diễn giống như cây nhị phân?
 - Mỗi nút sẽ chứa giá trị và **các** con trỏ trỏ đến các nút con của nó?
 - Bao nhiêu con trỏ cho một nút? >>Không hợp lý
- Mỗi nút sẽ chứa giá trị và **một** con trỏ trỏ đến một “**tập**” các nút con
 - Xây dựng “tập” như thế nào?

Biểu diễn cây tổng quát

- Sử dụng con trỏ nhưng mở rộng hơn:
 - Mỗi nút sẽ có 2 con trỏ: một con trỏ trỏ đến **nút con đầu tiên** của nó, con trỏ kia trỏ đến **nút anh em kề với nó**
 - Cách này cho phép quản lý số lượng tùy ý của các nút con





3.2. Duyệt cây tổng quát

1. Thứ tự trước:

- 1.Thăm gốc
- 2.Duyệt cây con thứ nhất theo thứ tự trước
- 3.Duyệt các cây con còn lại theo thứ tự trước

2. Thứ tự giữa

- 1.Duyệt cây con thứ nhất theo thứ tự giữa
- 2.Thăm gốc
- 3.Duyệt các cây con còn lại theo thứ tự giữa

3. Thứ tự sau:

- 1.Duyệt cây con thứ nhất theo thứ tự sau
- 2.Duyệt các cây con còn lại theo thứ tự sau
- 3.Thăm gốc

4. Ứng dụng của cây nhị phân

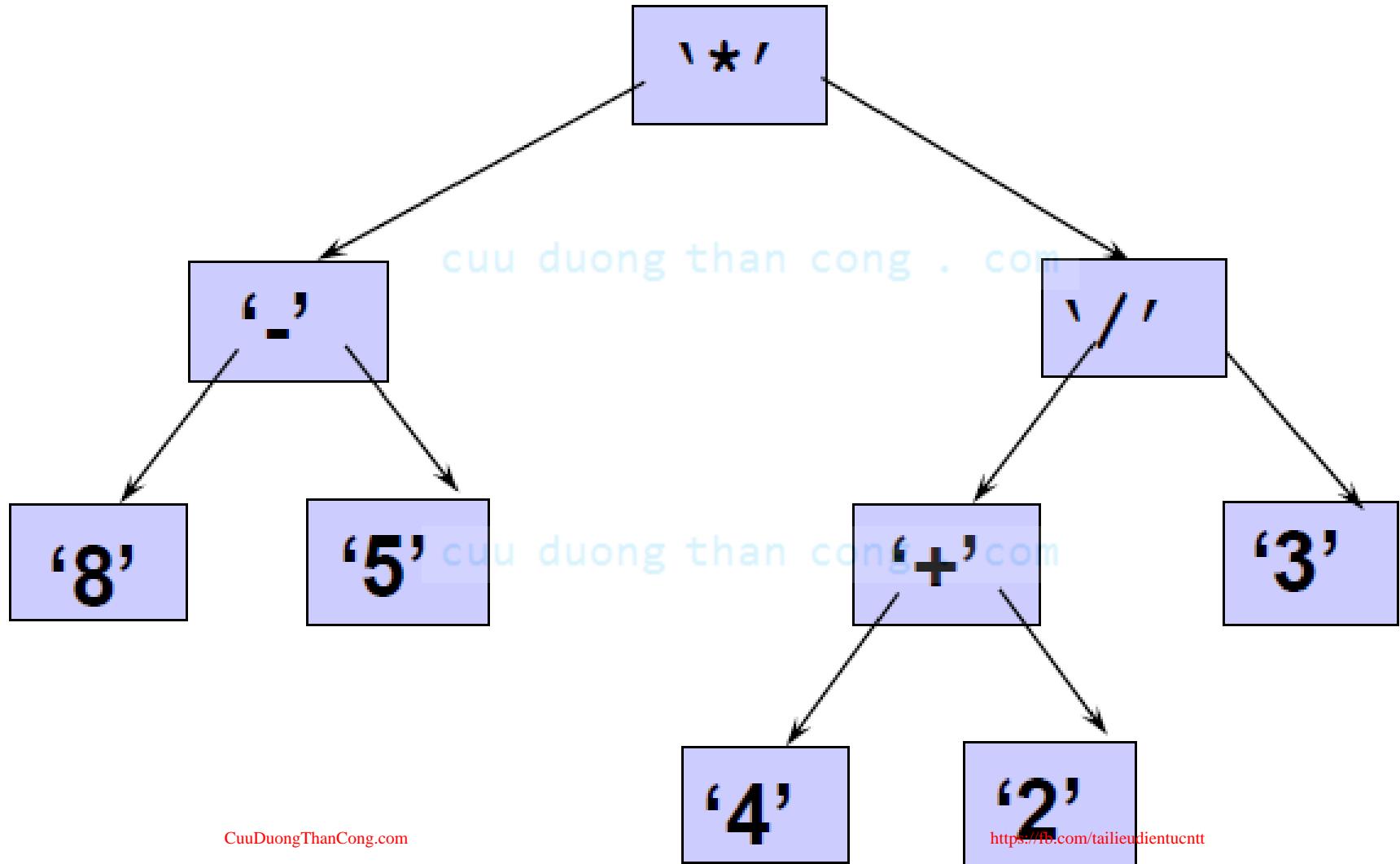
- Cây biểu diễn biểu thức
 – Tính giá trị biểu thức
 – Tính đạo hàm
- Cây quyết định

Cây biểu diễn biểu thức là.

Một loại cây nhị phân đặc biệt, trong đó:

- 1. Mỗi nút lá chứa một toán hạng**
- 2. Mỗi nút giữa chứa một toán tử**
- 3. Cây con trái và phải của một nút toán tử thể hiện các biểu thức con cần được đánh giá trước khi thực hiện toán tử tại nút gốc**

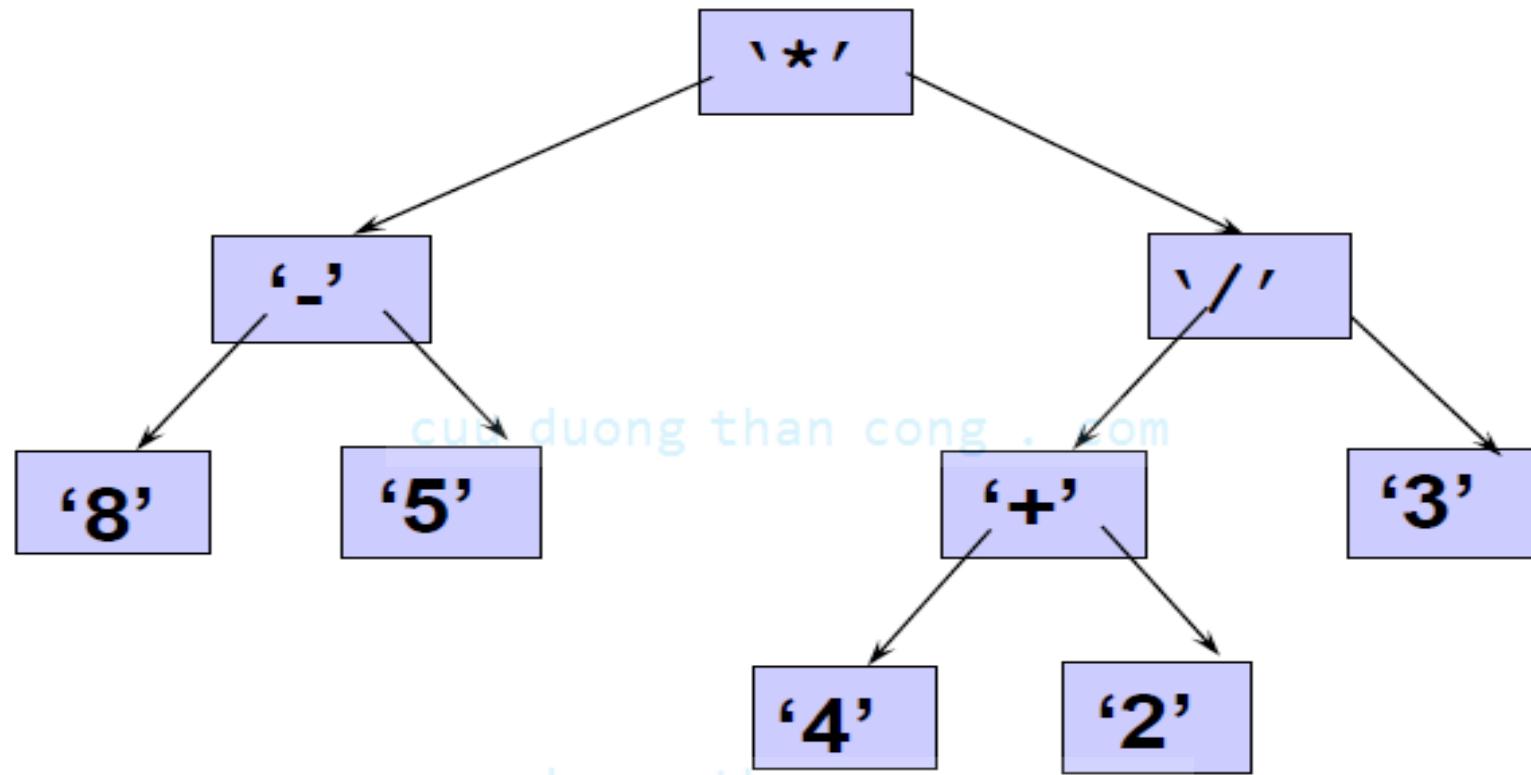
Biểu thức nhị phân



Các mức chỉ ra thứ tự ưu tiên

- Các mức (độ sâu) của các nút chỉ ra thứ tự ưu tiên tương đối của chúng trong biểu thức (không cần dùng ngoặc để thể hiện thứ tự ưu tiên).
- Các phép toán tại mức cao hơn sẽ được tính sau các phép toán có mức thấp.
- Phép toán tại gốc luôn được thực hiện cuối cùng.

- Dễ dàng để tạo ra các biểu thức tiền tố, trung tố, hậu tố

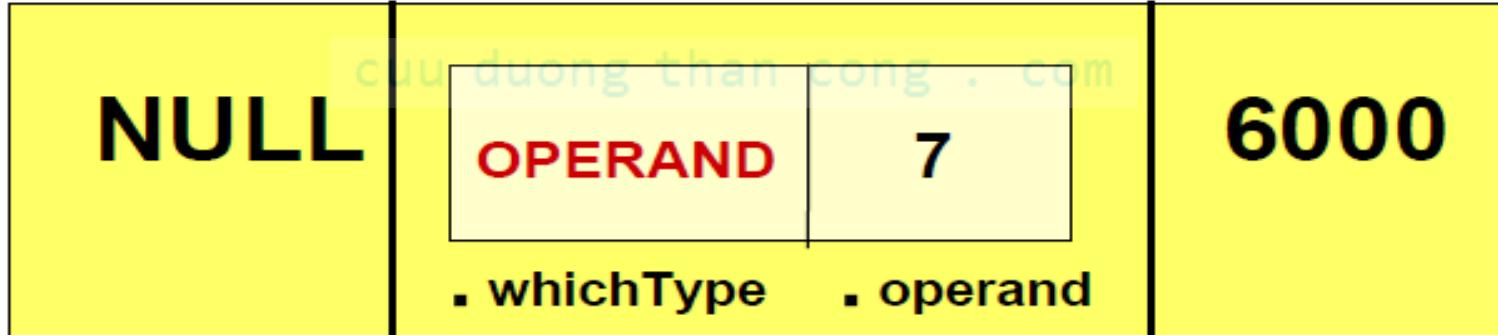


- Trung tố:** $((8 - 5) * ((4 + 2) / 3))$
- Tiền tố:** $* -8 5 / + 4 2 3$
- Hậu tố:** $8 5 -4 2 + 3 / *$

Cài đặt cây biểu thức

- Mỗi nút có 2 con trỏ

```
struct TreeNode {  
    InfoNode info ;// Dữ liệu  
    TreeNode *left ;// Trỏ tới nút con trái  
    TreeNode *right ; // Trỏ tới nút con phải  
};
```



- InfoNode có 2 dạng
- ```
enum OpType { OPERATOR, OPERAND } ;
struct InfoNode {
 OpType whichType;
 union // ANONYMOUS union
 {
 char operator;
 int operand ;
 }
};
```

**OPERATOR**

‘+’

• **whichType**      • **operation**

**OPERAND**

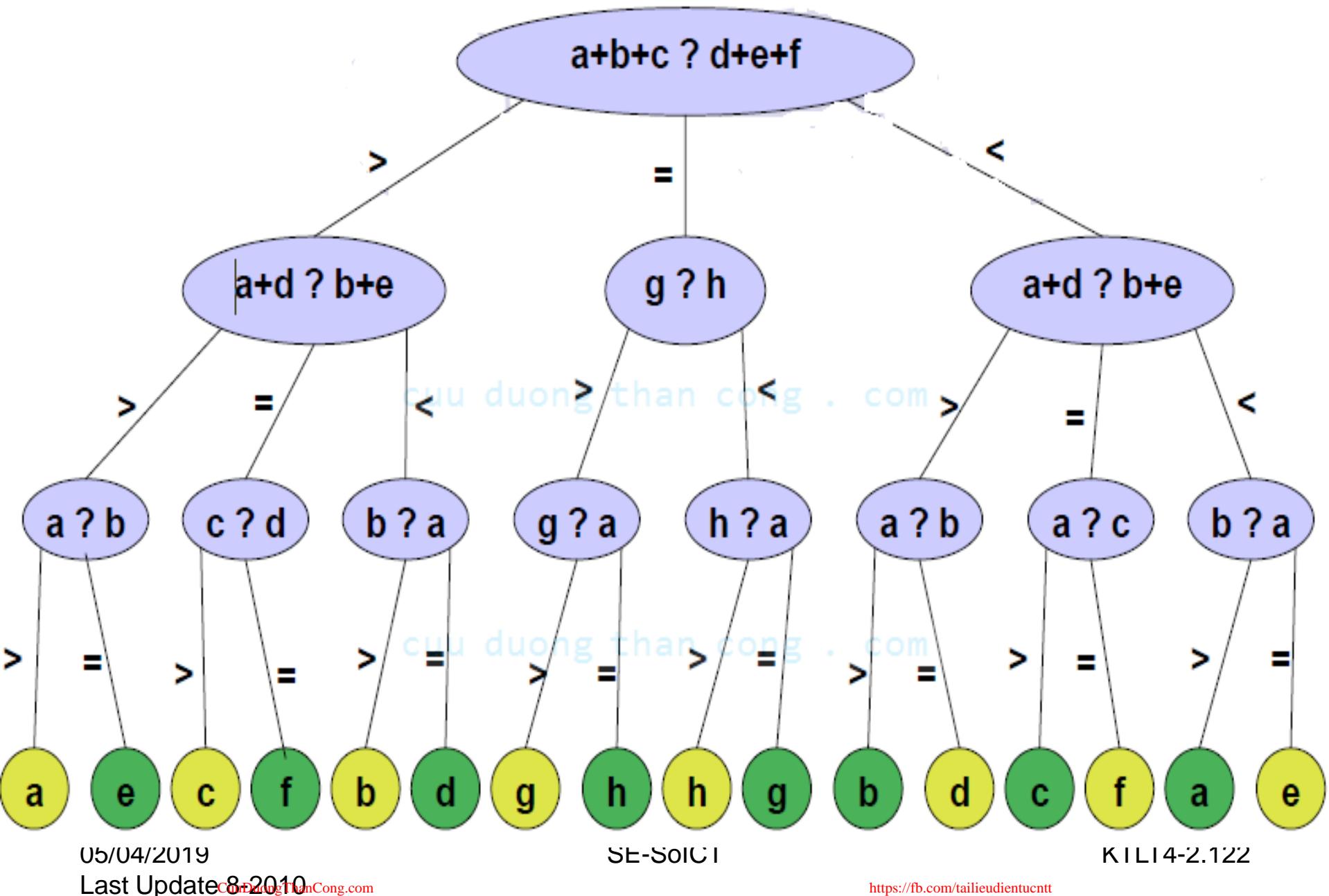
7

• **whichType**      • **operand**

```
int Eval(TreeNode* ptr){
 switch(ptr->info.whichType) {
 case OPERAND :
 returnptr->info.operand ;
 case OPERATOR :
 switch (tree->info.operation){
 case '+':
 return (Eval(ptr->left) + Eval(ptr->right)) ;
 case '-':
 return (Eval(ptr->left) -Eval(ptr->right)) ;
 case '*':
 return (Eval(ptr->left) * Eval(ptr->right)) ;
 case '/':
 return (Eval(ptr->left) / Eval(ptr->right)) ;
 }
 }
}
```

c)

- Dùng để biểu diễn lời giải của bài toán cần quyết định lựa chọn
- Bài toán 8 đồng tiền vàng:
  - Có 8 đồng tiền vàng a, b, c, d, e, f, g, h
  - Có một đồng có trọng lượng không chuẩn
  - Sử dụng một cân Roberval (2 đĩa)
  - Output:
    - Đồng tiền k chuẩn là nặng hơn hay nhẹ hơn
    - Số phép cân là ít nhất



```
void EightCoins(a, b, c, d, e, f, g, h) {
 if (a+b+c == d+e+f) {
 if (g > h) Compare(g, h, a);
 else Compare(h, g, a);
 }
 else if (a+b+c > d+e+f){
 if (a+d == b+e) Compare(c, f, a);
 else if (a+d > b+e) Compare(a, e, b);
 else Compare(b, d, a);
 }
 else{
 if (a+d == b+e) Compare(f,c,a);
 else if (a+d > b+e) Compare(d, b, a);
 else Compare(e, a, b);
 }
}
// so sánh x với đồng tiền chuẩn z
void Compare(x,y,z){
 if(x>y) printf("x nặng");
 else printf("y nhẹ");
}
```

05/04/2019

Last Update 8-2010

SE-SolCT

KLT4-2.123

<https://fb.com/tailieudientucntt>

# Lập trình phòng thủ

## <Defensive Programming>

(3LT-2BT)

cuu duong than cong . com

SE-SolCT

Last update 09-2010  
CuuDuongThanCong.com

<https://fb.com/tailieudientucntt>

KTLT 5.1

# DEFENSIVE PROGRAMMING?

- Xuất phát từ khái niệm defensive driving.
- Khi lái xe bạn luôn phải tâm niệm rằng bạn không bao giờ biết chắc được người lái xe khác sẽ làm gì. Bằng cách đó, bạn có thể chắc chắn rằng khi họ làm điều gì nguy hiểm, thì bạn sẽ không bị ảnh hưởng ( tai nạn).
- Bạn có trách nhiệm bảo vệ bản thân, ngay cả khi người khác có lỗi
- Trong defensive programming, ý tưởng chính là nếu chương trình (con) được truyền dữ liệu tồi, nó cũng không sao, kể cả khi với CT khác thì sẽ bị fault.
- Một cách tổng quát, lập trình phòng thủ nghĩa là : làm thế nào để tự bảo vệ mình khỏi thế giới

m'

SE-SolCT

KTLT 5.2

# 1. Bảo vệ CT khỏi các Invalid Inputs

- Trong thực tiễn :“Garbage in, garbage out.”
- Trong lập trình “rác rưởi vào – rác rưởi ra” là điều không chấp nhận
- 

o là gì !

- Với 1 CT tốt thì :"rác rưởi vào, không có gì ra", "rác rưởi vào, có thông báo lỗi" hoặc "không cho phép rác rưởi vào".
- Theo tiêu chuẩn ngày nay, "garbage in, garbage out" là dấu hiệu của những CT tồi, không an toàn

# Ba cách để xử lý rác vào

- **Kiểm tra giá trị của mọi dữ liệu từ nguồn bên ngoài**
  - Khi nhận dữ liệu từ file, bàn phím, mạng, hoặc từ các nguồn ngoài khác, hãy kiểm tra để đảm bảo rằng dữ liệu nằm trong giới hạn cho phép.
  - Hãy đảm bảo rằng giá trị số nằm trong dung sai và xâu phải đủ ngắn để xử lý. Nếu một chuỗi được dự định để đại diện cho một phạm vi giới hạn của các giá trị (như một i từ chối .

—

cuu duong than cong . com

:làm tràn bộ nhớ, injected SQL commands, injected ng  
html hay XML code, tràn số ...

# Ba cách để xử lý rác vào

- ***Check the values of all routine input parameters***
  - Kiểm tra giá trị của tất cả các tham số truyền vào các ~~hàm~~ ~~để~~ ~~đúng~~ ~~hàm~~ ~~đúng~~ cần như kiểm tra dữ liệu nhập từ nguồn ngoài khác
- ***Decide how to handle bad inputs***
  - Khi phát hiện 1 tham số hay 1 dữ liệu không hợp lệ, bạn ~~cần~~ ~~làm~~ ~~gi~~ ~~v~~ ~~ới~~ ~~n~~ ~~ó~~? Tùy thuộc tình huống, bạn có thể chọn 1 trong các phương án được mô tả chi tiết ở các phần sau.

## 2 Assertions

- 1 macro hay 1 CT con dùng trong quá trình phát triển ứng dụng, cho phép CT tự kiểm tra khi chạy.
- Return true >> OK, false >> có 1 lỗi gì đó trong CT.  
*cuu duong than cong . com*
- VD : Nếu hệ thống cho rằng file dữ liệu về khách hàng không bao giờ vượt quá 50 000 bản ghi, CT có thể chứa 1 assertion rằng số bản ghi là  $\leq 50\ 000$ . Khi mà số  
*cuu duong than cong . com*  
i trong CT.

- Sử dụng assertions để ghi lại những giả thiết được đưa ra trong code và để loại bỏ những điều kiện không mong đợi. Assertions có thể đc dùng để kiểm tra các giả thiết như :
  - Các tham số đầu vào nằm trong phạm vi mong đợi (tương tự với các tham số đầu ra)
  - file hay stream đang được mở (hay đóng) khi 1 CTC bắt đầu thực hiện (hay kết thúc)
  - 1 file hay stream đang ở bản ghi đầu tiên (hay cuối cùng) khi 1 CTC bắt đầu ( hay kết thúc) thực hiện
  - 1 file hay stream được mở để đọc, để ghi, hay cả đọc và ghi
  - Giá trị của 1 tham số đầu vào là không thay đổi bởi 1 CTC
  - 1 pointer là non-NULL
  - 1 mảng đc truyền vào CTC có thể chứa ít nhất X phần tử
  - 1 bảng đã đc khởi tạo để chứa các giá trị thực
  - 1 danh sách là rỗng (hay đầy) lkhi 1 CTC bắt đầu (hay kết thúc) thực hiện

- i không cần thấy các thông báo của assertion ;
- c dùng trong quá trình phát triển hay bảo dưỡng ứng dụng.
- 

cuu duong than cong . com

Rất nhiều NNLT hỗ trợ assertions : C++, Java và Visual Basic.

Kể cả khi NNLT không hỗ trợ, thì cũng có thể dễ ràng xd

VD:

```
#define ASSERT(condition, message) {
 if (!(condition)) {
 fprintf(stderr, "Assertion %s failed: %s\n",
 condition, message);
 exit(EXIT_FAILURE);
 }
}
```

# Guidelines for Using Assertions

- **xử lý lỗi với những điều kiện ta chờ đợi sẽ xảy ra; Dùng assertions cho các ĐK không mong đợi ( không bao giờ xảy ra)**
  - - o
  - bugs trong CT..
  -
- **xử lý vào trong assertions**
  - Điều gì xảy ra khi ta turn off the assertions ?

# Guidelines for Using Assertions(tt)

- **Với các hệ thống lớn, assert, và sau đó xử lý lỗi**
  - Với 1 nguyên nhân gây lỗi xác định, hoặc là dùng assertion hoặc error-handling , nhưng không dùng cả 2. ?
  - Với các HT lớn, nhiều người cùng Pt và kéo dài 5-10 năm, hoặc ~~hơn~~ [nhiều](#) ~~thanh~~ công . com
  - Cả assertions và error handling code có thể đc dùng cho cùng 1 lỗi. Ví dụ trong source code cho Microsoft Word, những điều kiện luôn trả về true thì đc dùng assertion, nhưng đồng thời cũng đc xử lý.
  - Với các hệ thống ~~cực~~ [công](#) ~~lớn~~ (VD Word) , assertions là rất có lợi vì nó giúp loại bỏ rất nhiều lỗi trong quá trình PT HT

### 3. Kỹ thuật xử lý lỗi (Error Handling Techniques)

- Error handling dùng để xử lý các lỗi mà ta chờ đợi sẽ xảy ra
- Tùy theo tình huống cụ thể, ta có thể trả về 1 giá trị trung lập, thay thế đoạn tiếp theo của dữ liệu hợp lệ, trả về cùng giá trị như lần trước, thay

cuuduongthancong . com

o hay tắt máy.

## Chắc chắn thay vì chính xác

- Giả sử một ứng dụng hiển thị thông tin đồ họa trên màn hình. Một vài điểm ảnh ở góc tọa độ dưới bên phải hiển thị màu sai. Ngày tiếp theo, màn hình sẽ làm mới, lỗi không còn. Phương pháp xử lý lỗi tốt nhất là gì?
- Chính xác có nghĩa là không bao giờ gặp lại lỗi
- Chắc chắn có nghĩa là phần mềm luôn chạy thông, kể cả khi có lỗi
- Ưu tiên tính chắc chắn để có tính chính xác. Bất cứ kết quả nào đó bao giờ cũng thường là tốt hơn so với Shutdown. Thỉnh thoảng trong các trình xử lý văn bản hiển thị một phần của một dòng văn bản ở phía dưới màn hình. Khi đó ta muốn tắt CT ? Chỉ cần nhấn trang lên hoặc xuống trang, màn hình sẽ làm mới, và sẽ hiển thị sẽ trở lại bình thường.
- Đôi khi, để loại bỏ 1 lỗi nhỏ, lại rất tốn kém. Nếu lỗi đó chắc chắn không ảnh hưởng đến mục đích cơ bản của ứng dụng, không làm CT bị die, hoặc làm sai lệch kết quả chính, người ta có thể bỏ qua, mà không cố sửa để có thể gặp phải các nguy cơ khác.
- Hiện tại có 1 dạng phần mềm “ chịu lỗi “ tức là loại phần mềm sống chung với lỗi, để đảm bảo tính liên tục, ổn định ...

## 4. Xử lý ngoại lệ - Exceptions

- Exception : bắt các tình huống bất thường và phục hồi chúng về trạng thái trước đó.
- Dùng Ngoại lệ để thông báo cho các bộ phận khác của chương trình về lỗi không nên bỏ qua
  - Lợi ích của ngoại lệ là khả năng báo hiệu điều kiện lỗi . Phương pháp tiếp cận khác để xử lý các lỗi tạo ra khả năng mà một điều kiện lỗi có thể truyền bá thông qua một cơ sở mã không bị phát hiện. Ngoại lệ có thể loại trừ khả năng đó.
- Chỉ dùng ngoại lệ cho những điều kiện thực sự ngoại lệ
  - Exceptions đc dùng trong những tình huống giống assertions— cho các sự kiện không thường xuyên, nhưng có thể không bao giờ xảy ra
  - Exception có thể bị lạm dụng và phá vở các cấu trúc, điều này dễ gây ra lỗi, vì làm sai lệch luồng điều khiển

- Ví dụ : trong các PM ứng dụng, khi xử lý dữ liệu ... ( C#)

```
try
{
 cmd.ExecuteNonQuery();
 ErrorsManager.SetError(ErrorIDs.KhongCoLoi);
}
catch
{
 ErrorsManager.SetError(ErrorIDs.SQLThatBai,
 database.DbName, "ten_strore");
}
```

VB.NET

```
Try
 Return CBO.FillCollection(CType(SqlHelper.ExecuteReader(ConStr,
 "TimHDOn", iSoHoaDon), IDataReader), GetType(ThanhToan.ChiTietHDInfo))
Catch ex As Exception
 mesagebox.show(ex.message)
End Try
```

```
Dim tran As SqlTransaction
```

```
Try
```

```
 conn.Open()
```

```
 tran = conn.BeginTransaction()
```

```
 SqlHelper.ExecuteNonQuery(tran, "ThemHDon",_
 HDInfo.SoHoaDonTC, HDInfo.TenKhach,_
 HDInfo.PhuongThucTT)
```

```
 iMaHD = GetMaHoaDon_Integer(tran)
```

```
 For Each objCT In arrDSCT
```

```
 SqlHelper.ExecuteNonQuery(tran, "ThemChiTietHD",_
 objCT.ChiTiet,_
 objCT.SoTienVND, iMaHDP)
```

```
 Next
```

```
 tran.Commit()
```

```
Catch ex As Exception
```

```
 tran.Rollback()
```

```
End Try
```

- Phục hồi tài nguyên khi xảy ra lỗi ?
  - Thường thì không phục hồi tài nguyên
  - Nhưng sẽ hữu ích khi thực hiện các công việc nhằm đảm bảo cho thông tin ở trạng thái rõ ràng và vô hại nhất có thể
  - Nếu các biến vẫn còn đc truy xuất thì chúng nên đc gán các giá trị hợp lý
  - Trường hợp thực thi việc cập nhật dữ liệu, nhất là trong 1 phiên – transaction – liên quan tới nhiều bảng chính, phụ, thì việc khôi phục khi có ngoại lệ là vô cùng cần thiết. ( rollback )

# 5.Gõ rối – debugging

- Các chương trình đã viết có thể có nhiều lỗi ? – tại sao phần mềm lại phức tạp vậy ?
- Sự phức tạp của CT liên quan đến cách thức tương tác của các thành phần của CT đó, mà 1 phần mềm lại bao gồm nhiều thành phần và các tương tác giữa chúng
- Nhiều kỹ thuật làm giảm số lượng các thành phần tương tác :
  - Che giấu thông tin
  - Trừu tượng hóa ...
- Có các kỹ thuật nhằm đảm bảo tính toàn vẹn thiết kế phần mềm
  - Documentation
  - Lập mô hình
  - Phân tích các yêu cầu
  - Kiểm tra hình thức
- Nhưng chưa có 1 kỹ thuật nào làm thay đổi cách thức xây dựng phần mềm => luôn xuất hiện lỗi khi test, phải loại bỏ = gõ rối !

- LTV chuyên nghiệp cũng tốn nhiều thời gian cho gỡ rối !
- Luôn rút kinh nghiệm từ các lỗi trước đó
- Viết code và gây lỗi là điều bình thường – vẫn đề làm sao để không lặp lại!
- LTV giỏi là người giỏi gỡ rối
- Gỡ rối không đơn giản, tốn thời gian => cần tránh gây ra lỗi. Các cách làm giảm thời gian gỡ rối là :
  - Thiết kế tốt
  - Phong cách LT tốt
  - Kiểm tra các ĐK biên
  - Kiểm tra các “khẳng định” – assertion và tính đúng đắn trong mã nguồn
  - Thiết kế giao tiếp tốt, giới hạn việc sử dụng dữ liệu toàn cục
  - Dùng các công cụ kiểm tra
- Phòng bệnh hơn chữa bệnh !!

- Động lực chính cho việc cải tiến các ngôn ngữ LT là cố gắng ngăn chặn các lỗi thông qua các đặc trưng ngôn ngữ như :
  - Kiểm tra các giới hạn biên của các chỉ số
  - Hạn chế không dùng con trỏ, bộ dọn dẹp, các kiểu dữ liệu chuỗi
  - Xác định kiểu nhập/xuất
  - Kiểm tra dữ liệu chặt chẽ.
- Mỗi ngôn ngữ cũng có những đặc tính dễ gây lỗi : lệnh goto, biến toàn cục, con trỏ trỏ tới vùng không xác định, chuyển kiểu tự động...
- LTV cần biết trước những đặc thù để tránh các lỗi tiềm ẩn, đồng thời cần kích hoạt mọi khả năng kiểm tra của trình biên dịch và quan tâm đến các cảnh báo
- Ví dụ : so sánh C,Pascal, VB ...

# Debugging Heuristic

| Debugging Heuristic           | When Applicable |
|-------------------------------|-----------------|
| (1) Understand error messages | Build-time      |
| (2) Think before writing      |                 |
| (3) Look for familiar bugs    |                 |
| (4) Divide and conquer        | Run-time        |
| (5) Add more internal tests   |                 |
| (6) Display output            |                 |
| (7) Use a debugger            |                 |
| (8) Focus on recent changes   |                 |

# Understand Error Messages

Gỡ rối khi **build-time** dễ hơn lúc **run-time**,  
khi và chỉ khi ta...

## (1) Hiểu đc các thông báo lỗi!!!

- 

### (preprocessor)

```
#include <stdioo.h>
int main(void)
/* Print "hello, world" to stdout and
 return 0.
{
 printf("hello, world\n");
 return 0;
}
```

Misspelled #include file

Missing \*/

```
$ gcc217 hello.c -o hello
hello.c:1:20: stdioo.h: No such file or directory
hello.c:3:1: unterminated comment
hello.c:2: error: syntax error at end of input
```

KLT 5.22

# Understand Error Messages (tt)

## (1) Hiểu đc các thông báo lỗi!!!

- ch (**compiler**)

```
#include <stdio.h>
int main(void)
/* Print "hello, world" to stdout and
 return 0. */
{
 printf("hello, world\n")
 retun 0;
}
```

Misspelled keyword

```
$ gcc217 hello.c -o hello
hello.c: In function `main':
hello.c:7: error: `retun' undeclared (first use in this function)
hello.c:7: error: (Each undeclared identifier is reported only once
hello.c:7: error: for each function it appears in.)
hello.c:7: error: syntax error before numeric constant
```

# Understand Error Messages (tt)

## (1) Hiểu đc các thông báo lỗi!!!

- 

m (**linker**)

```
#include <stdio.h>
int main(void)
/* Print "hello, world" to stdout and
 return 0. */
{
 printf("hello, world\n")
 return 0;
}
```

Misspelled  
function name

Compiler **warning** (not **error**):  
printf() is called before declared

Linker error: Cannot find  
definition of printf()

```
$ gcc217 hello.c -o hello
hello.c: In function `main':
hello.c:6: warning: implicit declaration of function `printf'
/tmp/cc43ebjk.o(.text+0x25): In function `main':
: undefined reference to `printf'
collect2: ld returned 1 exit status
```

TLT 5.24

# Các phương pháp gõ rối

- **Trình gõ rối :**

- IDE : kết hợp soạn thảo, biên dịch, gõ rối ...
- **Chạy từng bước** – step by step:   
họa cho phép chạy chương trình từng bước qua từng lệnh hoặc từng hàm, dừng ở những dòng lệnh đặc biệt hay khi xuất hiện những đk đặc biệt, bên cạnh đó có các công cụ cho phép định dạng và hiển thị giá trị các biến, biểu thức
- Trình gõ rối có thể đc kích hoạt trực tiếp khi có lỗi.
- Thường để tìm ra lỗi, ta phải xem xét thứ tự các hàm đã đc kích hoạt (theo vết) và hiển thị các giá trị các biến liên quan
- **Có nhiều công cụ gõ rối mạnh và hiệu quả**, tại sao ta vẫn mất nhiều thời gian và trí lực để gõ rối ?
- Nhiều khi các công cụ không thể giúp dễ ràng tìm lỗi, nếu đưa ra 1 câu hỏi sai, trình gõ rối sẽ cho 1 câu trả lời, nhưng ta có thể không biết là nó đang bị sai

# Các phương pháp gõ rối

- **Có đầu mối , phát hiện dễ ràng (Good clues, easy bugs) :**
  - trình dịch, thư viện hay bất cứ nguyên nhân nào khác ...Tuy nhiên, cuối cùng thì lỗi vẫn thuộc về CT.
  - Rất may là hầu hết các lỗi thường đơn giản và dễ tìm, Hãy khảo sát các đầu mối của việc xuất ra kq có lỗi và cố gắng suy ra nguyên nhân gây ra nó
  - o?
  - Suy luận ngược trở lại trạng thái của CT bị hỏng để xđ nguyên nhân gây ra lỗi
  - Gõ rối liên quan đến việc lập luận lùi, giống như tìm kiếm các bí mật của 1 vụ án. ~~để 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 đc vấn đề cần sửa và có thể phát hiện ra 1 số vđề khác~~

# Các phương pháp gõ rối

- **Tìm các lỗi tương tự :**

- Khi gặp vđè, hãy liên tưởng đến những trường hợp tương tự đã gặp.
- Vd1 : int n; scanf("%d",n); ? 

Int n;  
scanf("%d",&n)
- Vd2 : int n=1; double d=PI;  
  
printf("%d %f \n",d,n); ??
- Không khởi tạo biến (với C) cũng sẽ gây ra những lỗi khó lường.

# Các phương pháp gỡ rối

- **Kiểm tra sự thay đổi mới nhất**
  - Lỗi thường xảy ra ở những đoạn CT mới đc bổ sung
  - Nếu phiên bản cũ OK, phiên bản mới có lỗi => lỗi chắc chắn nằm ở những đoạn CT mới
  - Lưu ý, khi sửa đổi, nâng cấp : hãy giữ lại phiên bản cũ – đơn giản là comment lại đoạn mã cũ
  - Đặc biệt, với các hệ thống lớn, làm việc nhóm thì việc sử dụng các hệ thống quản lý phiên bản mã nguồn và các cơ chế lưu lại quá trình sửa đổi là vô cùng hữu ích ( source safe )

# Các phương pháp gỡ rối

- Tránh mắc cùng 1 lỗi 2 lần : Sau khi sửa 1 lỗi, hãy suy nghĩ xem có lỗi tương tự ở nơi nào khác không. VD :

```
for (i=1;i<argc;i++) {
 if (argv[i][0] !='-')
 break;
 switch (argv[i][1]) {
 case 'o' : /* tên tệp output*/
 outname = argv[i]; break;
 case 'f' :
 from = atoi(argv[i]); break;
 case 't' :
 to = atoi(argv[i]); break; }
}
```

p sai vì luôn có -o ở trước tên =>gp: outname= &argv[i][2];

Tương tự ?

Chú ý : nếu đơn giản có thể viết code khi ngủ thì cũng đừng ngủ gật khi viết code.

# Các phương pháp gõ rối

- **t (stack trace)**

int arr[N];

qsort (arr, N, sizeof(arr[0]),icmp)

t VD sau:

t:

0. strcmp(0x1a2,0x1c2)[“strcmp, s”:31]

1. scmp (p1=0x10001048,p2=0x1000105c)[badqs.c”:13]

2. qst(0x 10001048, 0x 10001074,0 x 400b20,0x4) [“qsort.c”:147]

....

5. \_istart () [“crt...s”:13]

strcmp...

SE-SolCT

KTLT 5.30

# Các phương pháp gỡ rối

- **Gỡ rối ngay khi gặp**

- Khi phát hiện lỗi, hãy sửa ngay, đừng để sau mới sửa, vì có thể lỗi không xuất hiện lại ( do tình huống... )  
*cuu duong than cong . com*
- Cũng đừng quá vội vàng, không suy nghĩ chín chắn, kỹ càng, vì có thể việc sửa chữa này ảnh hưởng tới các tình huống khác  
*cuu duong than cong . com*

# Các phương pháp gõ rối

- **Đọc trước khi gõ vào**
  - **Đừng vội vàng, khi không rõ điều gì thực sự gây ra lỗi và sửa không đúng chỗ sẽ có nguy cơ gây ra lỗi khác**
  - **Có thể viết đoạn code gây lỗi ra giấy=> tạo cách nhìn khác, và tạo cơ hội để nghĩ suy**
  - **Đừng miên man chép cả đoạn không có nguy cơ gây lỗi, hoặc in toàn bộ code ra giấy in => phá vỡ cây cấu trúc**

# Các phương pháp gõ rối

- **Giải thích cho người khác về đoạn code**
  - Tạo đk để ngẫm nghĩ
  - Thậm chí có thể giải thích cho người không phải LTV
  - Extrem programming : làm việc theo cặp, pair programming, người này LT, người kia kiểm tra, và ngược lại
  - Khi gấp vấn đề, khó khăn, chậm tiến độ, lập tức thay đổi công việc => rút ra khỏi luồng quán tính sai lầm ...

(No clues, hard bugs)

- **Làm cho lỗi xuất hiện lại**
  - Cố gắng làm cho lỗi có thể xuất hiện lại khi cần
  - Nếu không dc, thì thử tìm nguyên nhân tại sao lại không dc
- **Chia để trị**
  - Thu hẹp phạm vi
  - Tập trung vào dữ liệu gây lỗi

## (No clues, hard bugs)

- **Hiển thị kết quả để định vị khu vực gây lỗi**
  - Thêm vào các dòng lệnh in giá trị các biến liên quan, hoặc đơn giản xác định tiến trình thực hiện: “đến đây 1” ...
- **Viết mã tự kiểm tra**
  - Viết thêm 1 hàm để kiểm tra, gắn vào trước và sau đoạn có nguy cơ, comment lại sau khi đã xử lý lỗi
- **Tạo log file**
- **Lưu vết**
  - Giúp ghi nhớ dc các vấn đề đã xảy ra, và giải quyết các vđề tương tự sau này, cũng như khi chuyển giao CT cho người khác..

# Hiển thị KQ ..

In các giá trị tại các điểm nhạy cảm

-Poor:

```
printf("%d", keyvariable);
```

stdout is buffered; CT  
có thể có lỗi trước khi  
hiển ra output

-Maybe better:

```
printf("%d\n", keyvariable);
```

In '\n' sẽ xóa bộ nhớ  
đệm stdout , nhưng sẽ  
không xóa khi in ra file

-Better:

```
printf("%d", keyvariable);
fflush(stdout);
```

Gọi fflush() để làm sạch  
buffer 1 cách tường  
minh

# Hiển thị KQ (cont.)

– Maybe even better:

```
fprintf(stderr, "%d", keyvariable);
```

In debugging output ra **stderr**; debugging output có thể tách biệt với các in ân của CT

– Maybe better still:

```
FILE *fp = fopen("logfile", "w");
...
fprintf(fp, "%d", keyvariable);
fflush(fp);
```

Ngoài ra: stderr không dùng buffer

Ghi ra 1 a log file

# Các phương pháp gõ rối

- **Phương sách cuối cùng (last resorts-p 128)**
  - Dùng 1 trình gõ rối để chạy từng bước
  - Nhiều khi vẫn đề tưởng quá đơn giản nhưng lại không phát c trưng NN.  
*( == và != nhỏ hơn !)*
  - Thú tự các đối số của lời gọi hàm : ví dụ : **strcpy(s1,s2)**  

```
int m[6]={1,2,3,4,5,6}, *p,*q;
p=m; q=p+2; *p++ =*q++; *p=*q; ???
```
  - Lỗi loại này khó tìm vì bản thân ý nghĩ của ta vạch ra 1 hướng suy nghĩ sai lệch : coi điều không đúng là đúng
  - Đôi khi lỗi là do nguyên nhân khách quan : Trình biên dịch, thư viện hay hệ điều hành, hoặc lỗi phần cứng : 1994 lỗi xử lý dấu chấm động trong bộ xử lý Pentium.

- **Các lỗi xuất hiện thất thường :**
  - Khó giải quyết
  - Thường gán cho lỗi của máy tính, hệ điều hành ...
  - Thực ra là do thông tin của chính CT : không phải do thuật toán, mà do thông tin bị thay đổi qua mỗi lần chạy
  - Các biến đã đc khởi tạo hết chưa ?
  - Lỗi cấp phát bộ nhớ ? Vd :

```
char *vd(char *s) {
 char m[101];
 strncpy(m,s,100)
 return m;
}
```
  - Giải phóng bộ nhớ động ?  

```
for (p=listp; p!=NULL; p=p->next) free(p) ; ???
```

i (p 131)

•

cuu duong than cong . com

cuu duong than cong . com

SE-SolCT

Last update 09-2010  
CuuDuongThanCong.com

<https://fb.com/tailieudientucntt>

KT LT 5.40

# Tóm lại

- Gỡ rối là 1 nghệ thuật mà ta phải luyện tập thường xuyên
- Nhưng đó là nghệ thuật mà ta không muốn
- Mã nguồn viết tốt có ít lỗi hơn và dễ tìm hơn
- Đầu tiên phải nghĩ đến nguồn gốc sinh ra lỗi
- Hãy suy nghĩ kỹ càng, có hệ thống để định vị khu vực gây lỗi
- Không gì bằng học từ chính lỗi của mình – điều này càng đúng đối với LTV

# Thêm – Những lỗi thường gặp với C, C++

1. Array as a parameter handled improperly – Tham số mảng đc xử lý không đúng cách
2. Array index out of bounds – Vượt ra ngoài phạm vi chỉ số mảng
3. Call-by-value used instead of call-by reference for function parameters to be modified – Gọi theo giá trị, thay vì gọi theo tham chiếu cho hàm để sửa
4. Comparison operators misused – Các toán tử so sánh bị dùng sai
5. Compound statement not used - Lệnh phức hợp không đc dùng
6. Dangling else - nhánh else khong hợp lệ
7. Division by zero attempted - Chia cho 0
8. Division using integers so quotient gets truncated – Dùng phép chia số nguyên nên phần thập phân bị cắt
9. Files not closed properly (buffer not flushed) - File không đc đóng phù hợp ( buffer không bị dẹp)
10. Infinite loop - lặp vô hạn
11. Global variables used – dùng biến tổng thể

12. IF-ELSE not used properly – dùng if-else không chuẩn
13. Left side of assignment not an L-value - phía trái phép gán không phải biến
14. Loop has no body – vòng lặp không có thân
15. Missing "&" or missing "const" with a call-by-reference function parameter – thiếu dấu & hay từ khóa const với lời gọi tham số hàm theo tham chiếu
16. Missing bracket for body of function or compound statement – Thiếu cặp {} cho thân của hàm hay nhóm lệnh
17. Mission reference to namespace - Thiếu tham chiếu tới tên miền
18. Missing return statement in a value-returning function – Thiếu return
19. Missing semi-colon in simple statement, function prototypes, struct definitions or class definitions – thiếu dấu ; trong lệnh đơn ...
20. Mismatched data types in expressions – kiểu dữ liệu không hợp
21. Operator precedence misunderstood - Hiểu sai thứ tự các phép toán

22. Off-by-one error in a loop – Thoát khỏi bởi 1 lỗi trong vòng lặp
23. Overused (overloaded) local variable names - Trùng tên biến cục bộ
24. Pointers not set properly or overwritten in error – Con trỏ không đc xác định đúng hoặc trỏ vào 1 vị trí không có
25. Return with value attempted in void function – trả về 1 giá trị trong 1 hàm void
26. Undeclared variable name – không khai báo biến
27. Un-initialized variables – Không khởi tạo giá trị
28. Unmatched parentheses – thiếu }
29. Un-terminated strings - xâu không kết thúc , thiếu “
30. Using "=" when "==" is intended or vice versa
31. Using "&" when "&&" is intended or vice versa
32. "while" used improperly instead of "if" – while đc dùng thay vì if