



ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Chương 1: Tổng quan về kỹ thuật lập trình

Mục tiêu môn học?

- Học phần *Kỹ thuật lập trình* trang bị cho sinh viên những kỹ thuật cơ bản nhất mà một lập trình viên chuyên nghiệp cần phải nắm vững để viết mã nguồn hiệu quả. Các kiến thức giảng dạy góp phần quan trọng giúp sinh viên phát triển được các ứng dụng phần mềm chất lượng cao trong thực tế.
- Học phần này trang bị cho sinh viên các kỹ thuật lập trình quan trọng như quản lý bộ nhớ, hàm, kỹ thuật đệ quy, kỹ thuật sử dụng các cấu trúc dữ liệu để giải quyết vấn đề, kỹ thuật viết mã nguồn hiệu quả, kỹ thuật lập trình phòng ngừa, kỹ thuật gỡ rối, tinh chỉnh mã nguồn, phong cách lập trình. Học phần có các buổi thực hành nhằm rèn luyện và nâng cao kỹ năng lập trình của sinh viên.

Tài liệu học tập

- [1] Bài giảng trên lớp
- [2] Trần Đan Thư (2014). Kỹ thuật lập trình. NXB Khoa học và kỹ thuật
- [3] McConnell, Steve (2004). Code Complete: A Practical Handbook of Software Construction, 2d Ed. Redmond, Wa.: Microsoft Press.
- [4] Kernighan & Plauger (1978). The elements of programming style. McGraw-Hill; 2nd edition
- [5] Brian W. Kernighan and Rob Pike (1999). The Practice of Programming. Addison-Wesley; 1st Edition
- [6] Nicolai M. Josuttis. The C++ Standard Library: A Tutorial and Reference (2nd Edition), 2012.

Đánh giá học phần

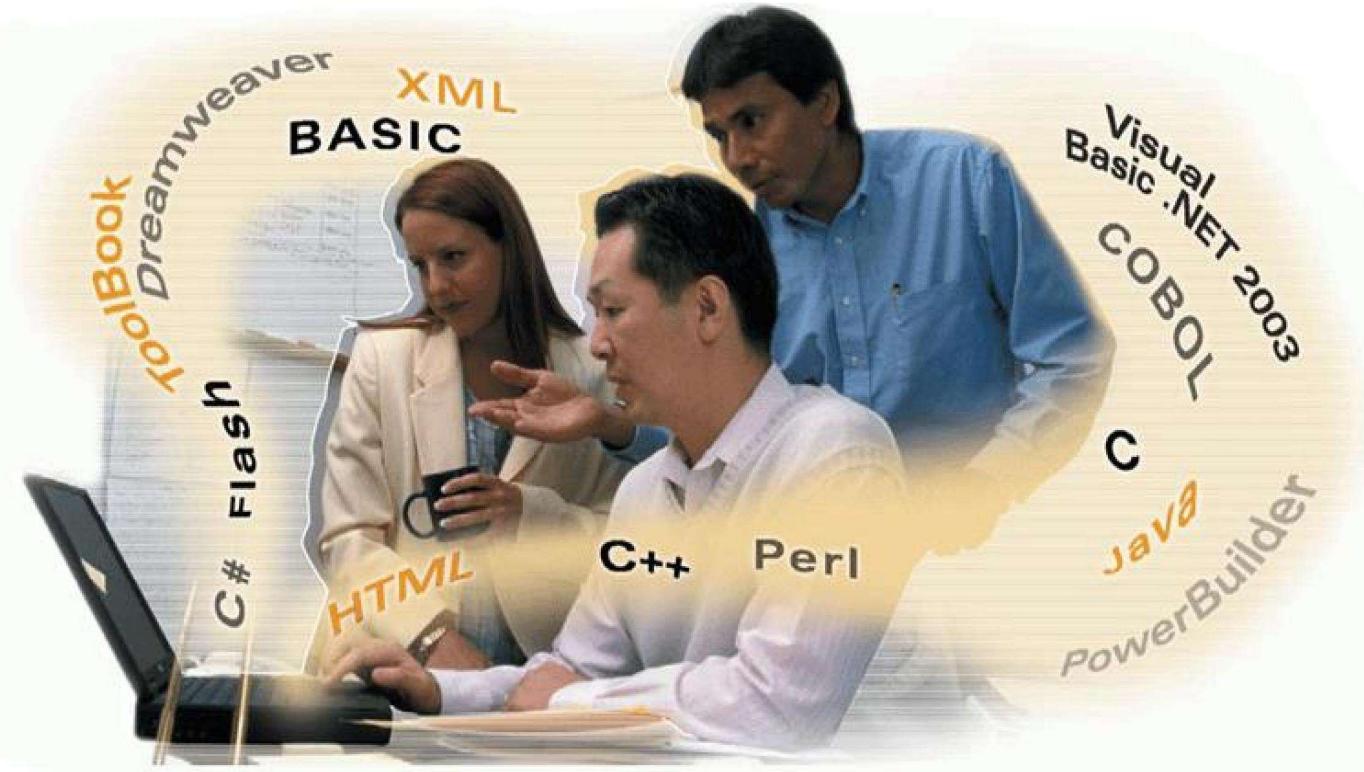
Điểm thành phần [1]	Phương pháp đánh giá cụ thể [2]	Mô tả [3]	CĐR được đánh giá [4]	Tỷ trọng [5]
A1. Điểm quá trình (*)	Đánh giá quá trình			40%
	A1.1. Bài tập về nhà	Tự luận M2.1 M2.2		10%
	A1.2a. Bài tập nhóm	Báo cáo M2.3 M1.4		30%
	A1.2b. Thi giữa kỳ	Tự luận và/ hoặc trắc nghiệm M2.1 M2.2 M2.3		30%
A2. Điểm cuối kỳ	A2.1. Thi cuối kỳ	Tự luận và/ hoặc trắc nghiệm M1.2 M1.4 M2.2 M2.3		60%

Tổng quan về lập trình

Hoạt động của chương trình máy tính và ngôn ngữ lập trình

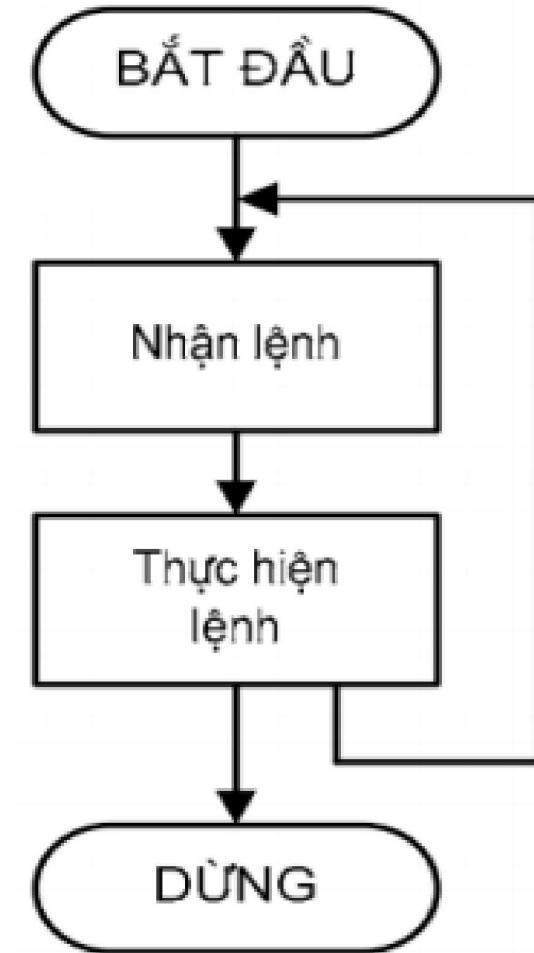
Chương trình máy tính và ngôn ngữ lập trình

- Chương trình máy tính: Tập hợp các lệnh chỉ dẫn cho máy tính thực hiện nhiệm vụ
- Ngôn ngữ lập trình: Dùng để viết các lệnh, chỉ thị



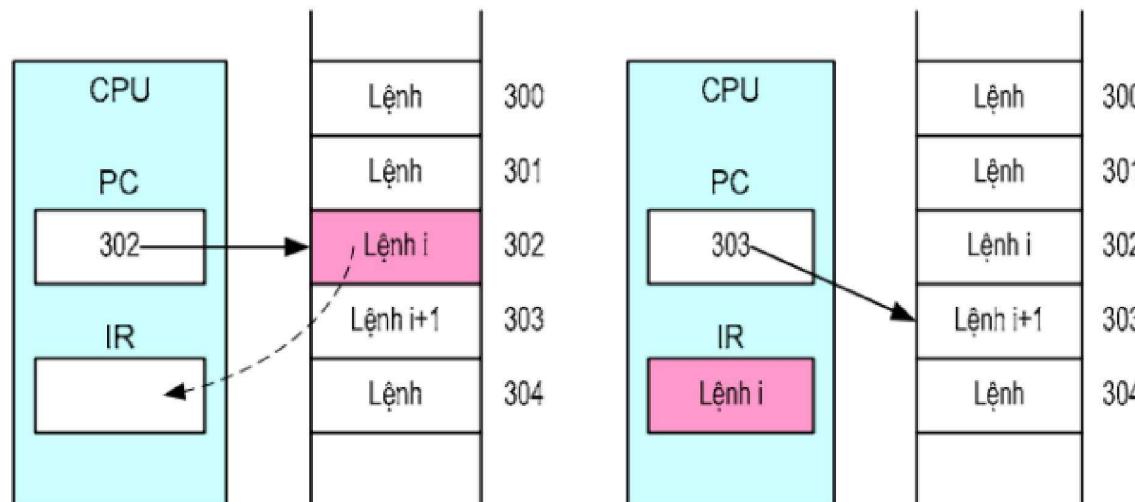
Hoạt động của chương trình máy tính

- Chương trình máy tính được nạp vào bộ nhớ chính (primary memory) như là một tập các lệnh viết bằng ngôn ngữ mà máy tính hiểu được, tức là một dãy tuần tự các số nhị phân (binary digits).
- Tại bất cứ một thời điểm nào, máy tính 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 được gọi là luồng điều khiển (flow of control).



Hoạt động của chương trình máy tính

- Bắt đầu mỗi chu trình lệnh, CPU nhận lệnh từ bộ nhớ chính.
 - PC (Program Counter): thanh ghi giữ địa chỉ của lệnh sẽ được nhận
 - Lệnh được nạp vào thanh ghi lệnh IR (Instruction Register)
- Sau khi lệnh được nhận vào, nội dung PC tự động tăng để trỏ sang lệnh kế tiếp



Ngôn ngữ lập trình

- Ngôn ngữ lập trình là một hệ thống các ký hiệu dùng để liên lạc, trao đổi với máy tính nhằm thực thi một nhiệm vụ tính toán.
- Có rất nhiều ngôn ngữ lập trình (khoảng hơn 1000), phần lớn là các ngôn ngữ hàn lâm, có mục đích riêng hay phạm vi.

Ngôn ngữ lập trình

Có 3 thành phần căn bản của bất cứ 1 NNLT nào:

- *Mô thức lập trình* là những nguyên tắc chung cơ bản, dùng bởi LTV để xây dựng chương trình.
- *Cú pháp* của ngôn ngữ là cách để xác định cái gì là hợp lệ trong cấu trúc các câu của ngôn ngữ; Nắm được cú pháp là cách để đọc và tạo ra các câu trong các ngôn ngữ tự nhiên, như tiếng Việt, tiếng Anh. Tuy nhiên điều đó không có nghĩa là nó giúp chúng ta hiểu hết ý nghĩa của câu văn.
- *Ngữ nghĩa* của 1 program trong ngôn ngữ ấy. Rõ ràng, nếu không có semantics, 1 NNLT sẽ chỉ là 1 mớ các câu văn vô nghĩa; như vậy semantics là 1 thành phần không thể thiếu của 1 ngôn ngữ.

Mã máy – Machine code

Máy tính chỉ nhận các tín hiệu điện tử - có, không có - tương ứng với các dòng bits.

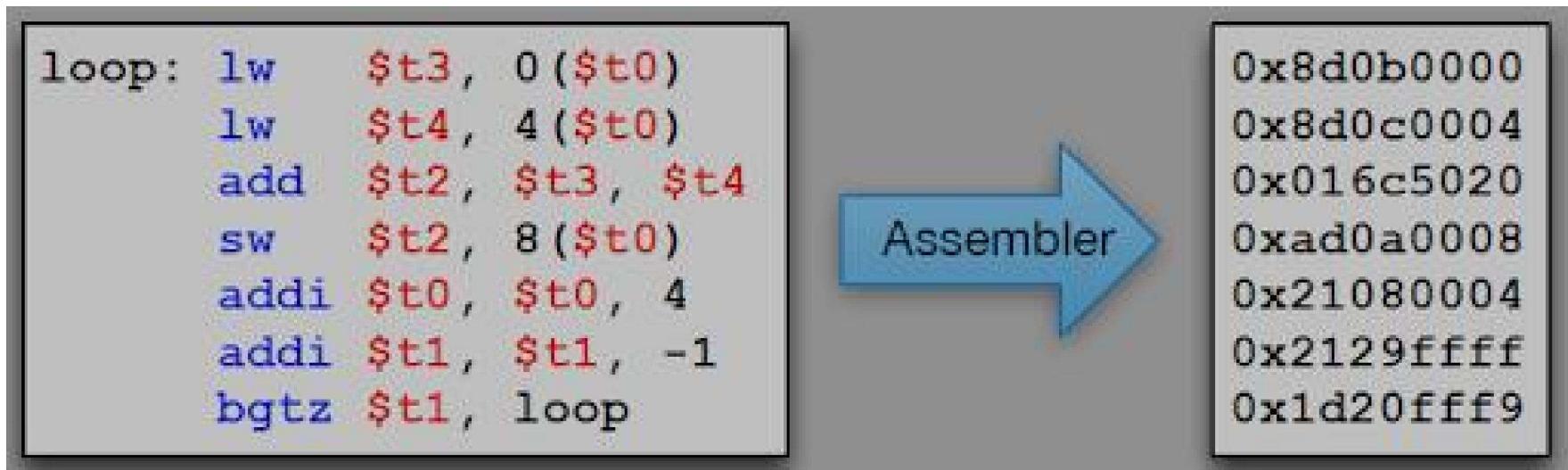
Một chương trình ở dạng đó gọi là mã máy (machine code).

9E4C1C	7F	4E9E	1C7A4E	4E7C5A	1C4E	4E4F	7C1C	9E	1C4E4D4C9E	7A7A3A5C	1C	4F	
1C	5B	7F	5E5B4C4C1C5B5A6D5E9E4C	4F9B7C	9E1C	9B4C5E	5A5B	1C9B7C9E	9E1C1C1C3B				
6C4D4E	7C	9B5B	5E9B	1C3A4E9B	3A	7F3A	6D4C	7F9E3B4D	9B3A7C	1C4C4C6D			
7C4F	9E	5E5E	4C4E	7A4C5C6C7C4C7C7A	7A	9C	9B	4D	5B5B	5B4E	4D5E4E		
4E	5C5E5A4C	6C5E7F	4D4E7A	4C9B9B3B	9B	6C4C9E5A7A5C5E5A	5B6D	9B9B7F5C9B					
5B	5A3B1C5E9E	5B7A	5B5A5A5A	5E	3A4C4C7A	5B7A9B	5E7C	7A4E7A	5C	4C5E			
5B6D	1C	7C4C9B	5E	7A3B	7C5E	1C5B	6D	5A1C	4D5A5B	7A	7F5B	6C7A	
3B6D	5B9E6D4E5E1C9E	1C9C1C5E4C	5E7C9B	9B5B4C	4E	5B	7A9E7A9E1C7F9E	4C5B4C					
3A	3B	4C9C5E7A4C1C5B6D4D5B1C5B	4C7C	5E4C1C1C5A4C4E	5E6C4D4C4C	4C	9E9B9E5E	1C					
5B	5A	3B4D	4C	6C7F1C	1C4C4E7C5E4C	9B	4D5B9B5E	1C7A4D4C5C7A5A7F	4C	9E			
7A5B	9C	4C	4C9E4E4D	9B	4E7A	1C7A7C9E	5A5B9B4E7C	7A	4C9C1C4E6C5B5E5A6C4C4C5E				
3B	9B4C9E	7F9B	3B	5E	4D	4D6D	1C9B	9B	5B5E9B5B5B	9B7F1C			
1C4D1C	6C5B1C	9E4F7F	3B	7A	5A4C3A	9E1C4E3B4E	9E7A7F	1C7C	3A9E	7F4C5E9E1C1C4C			
1C1C6C	4C	9E5E5C5B7F	7C	9E4E1C9E9B	7F	4C9B4F9B	3B3B9E6D7C3B4C5E3A	7A5B4C					
1C	5E7A7C4D	4D4C5C5B6D5A9E	6D7F	9E	3B	7C	4C4C4E4E7A1C	4C3A3B4E7C5E9E	6D				
5B	3B4C	3B	1C5E	7F3A	7F9B4C	4C5E4C	9B	7F	6D3B7F	4C	5E4C4C5E	6C3B9E5E	
4C6D	9B9E5E	4C1C5B5E	3A	5A7F	5A5E7A4D	7A	7C7F3B6C	5A6D1C1C3B4D1C	5E				
5B9E	5B4C9C9E5E	4D3B	6D	4E	1C7A1C	3B	6C	4E	5C5B4F9B	6C3A	5A	7F7A	
5C7A4C7A5B4E	7F	7A3B	9B4C	1C	7A9B5A4E9E	7C1C6D9B	7A3B9E5E4E7F4F3B4C4C1C						
5B4C4C7F	3B5B	5C6C5E9B7F	1C9B3B	4F	9B9B6D9B	3B6D4C3A	3B4C9E9B1C	7C5E	7F4C4C				
5B4D1C	5A4C7A5E7F	4E6D5A5E	5E7C	3B3B5B	7A6C	5E	7C1C4C4F4E4C6C						
4E	9E9B5A3B5B	9B9B1C1C5E4C5B	3A4D4F5A4F5E	4E	3B	5B3B	9C4C	7F9B4E					
1C	3B	7F5B5B7C9B4E	9B4C7A5E9B9B4D4E	7F6C	1C	7F	3B	5B	4E6C4E	5E4C7F	4D		
9B5B1C9E5E5E	4E4D4C6C5E9B	9C4C7C	4C7C9B	7F	4C9E	4F5E4C4E	5A	4E7F					
5B	5E4C7C7A	4E1C5B1C6D4C	7F3B4C	9B5B6D9B3A	9E4F	4E	4E5A5E	7C	5A	4C			
1C	9B5C	4E4C	7C	3B5B5B4D3B	9E9C7A4C	4C4C7C	9B9B4C4C	5B5B7F4D9B					
9B1C9B	4C	7C5B5A4E4C4E3B	5A	4F6C5E4E9B9E4F9C7F	3B7A4F	5A6C5E	5B1C	3A5C	7C7A7A				
4E7F	9B5C1C1C	4D5B5E7C7A4C	5E4C	5A9E	9E	6D3B9B	4E	6C	6C5E	3B5E7C9B	4C	4D5E	
5A4E	5B4C	7F	6D	1C7F	3A	6D	9E3A	4E	5C9B3B7A	5E9E4E4C	4C7F4C5E1C4C4C		
4C	6C9E	5C	7C4C	4C	7F1C9E	9C	5B6C4C4C1C4C4E5E3B	4C4E	5A4C5E6C5E7F7A	9B4E5A			
6D	3A1C	7F9E	5C	4C4C	3B4E7A3B	3A1C5E	7A3B4E4C5E	1C6D6D7F	5E	4C			
4D	4D	5E5B6C	7A4C6D1C	9E	4E	7F1C5B9B	7C	6C	5B	9B3B	4C5A4E4E5B5C5A	3A	
9B3A5A7A	9B1C3B	1C	7A	3B	5E	5A1C4F4F	1C6D9E	9B	9B7C4E1C4C5B9B9E	1C5E7A			
5E4E5E7F5B	6C	6C4C4C4C5E7C3A7F4C5E6C	5A	5A	9B	4F	4C4C5B4E4C1C5A5E	5E5A	1C7A1C				
9E	3A5E	6D4C	3B	4D4F	5B6C	4C7F6D	6C	5E	3A4C	5E	5A4C	4C	7C7A

Hợp ngữ - Assembly

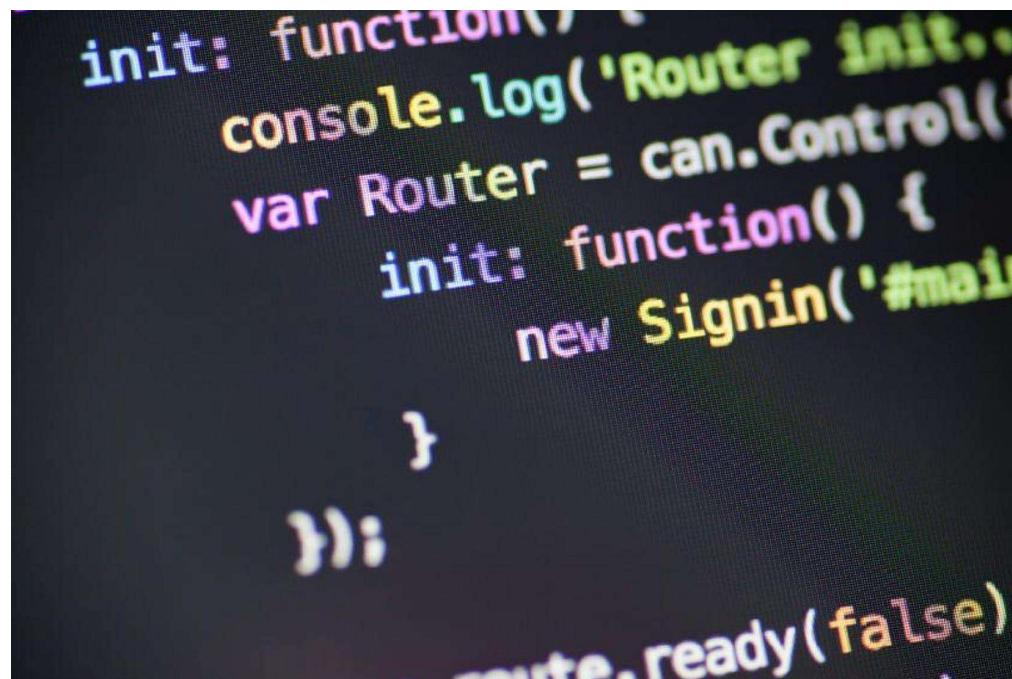
Là bước đầu tiên của việc xây dựng cơ chế viết chương trình tiện lợi hơn thông qua các ký hiệu, từ khóa và cả mã máy.

Tất nhiên, để chạy được các chương trình này thì phải chuyển thành machine code.



Ngôn ngữ lập trình bậc cao

- Thay vì dựa trên phần cứng (machine-oriented) cần tìm cơ chế dựa trên vấn đề (problem-oriented) để tạo chương trình
- Gần gũi với ngôn ngữ tự nhiên hơn, thường sử dụng các từ khóa giống tiếng Anh

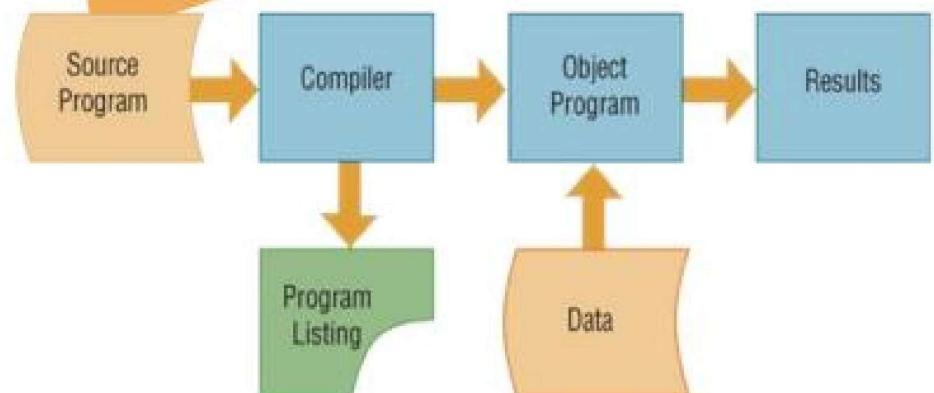


```
init: function(),
  console.log('Router init..',
  var Router = can.Control({
    init: function() {
      new Signin('#main');
    }
  });
  // route ready
  route.ready(false);
```

Trình dịch - compiler

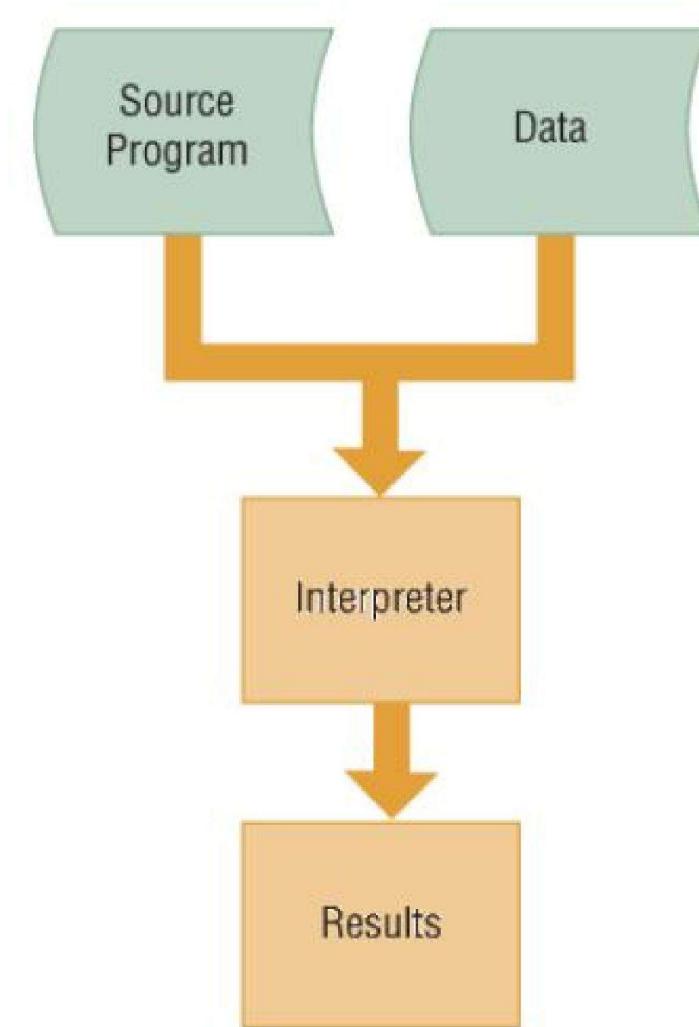
- Chương trình thực hiện biên dịch toàn bộ chương trình nguồn thành mã máy trước khi thực hiện

```
* 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.
```



Thông dịch - interpreter

- Chương trình dịch và thực hiện từng dòng lệnh của chương trình cùng lúc
- Dịch từ ngôn ngữ này sang ngôn ngữ khác, không tạo ra chương trình dạng mã máy hay assembly



Các mô thức lập trình

Programming paradigm

Các mô thức lập trình

- Imperative paradigm
- Functional paradigm
- Logical paradigm
- Object-oriented paradigm
- Visual paradigm
- Parallel paradigm
- Concurrent paradigm
- Distributed paradigm
- Service-oriented paradigm

Imperative paradigm – hướng mệnh lệnh

first do this and next do that

Thành phần:

- **Declarative statements**, các lệnh khai báo: cung cấp các tên cho biến. Các biến này có thể thay đổi giá trị trong quá trình thực hiện Chương trình.
- **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.
- **Module**: chia chương trình thành các chương trình con: Functions & Procedures

Functional paradigm – hướng chức năng

Thành phần

- Tập hợp các cấu trúc dữ liệu và các hàm liên quan
- Tập hợp các hàm cơ sở
- Tập hợp các toán tử

Đặc trưng cơ bản: *module hóa chương trình*

- Chức năng là biểu diễn của một biểu thức
- Giải thuật thực hiện theo từng bước
- Giá trị trả về là không thể biến đổi
- Không thể thay đổi CTDL của giá trị nhưng có thể sao chép các thành phần tạo nên giá trị đó
- Tính toán bằng cách gọi các chức năng

Functional paradigm – hướng chức năng

Ví dụ đoạn
chương trình
tìm đường đi
trên đồ thị
cho bởi danh
sách kề bằng
ngôn ngữ lập
trình hàm
Racket

```
(define graph '((A (C D E))
                (B (E J)))
                (C ()))
                (D (F J)))
                (E (K)))
                (F (K H)))
                (H ()))
                (J (H)))
                (K ()))))

;; neighbours: Node Graph --> (listof Node)
;;   requires: v is a node in g
(define (neighbours v g)
  (cond
    [(empty? g) (error "Node not found")]
    [(symbol=? v (first (first g))) (second (first g))]
    [else (neighbours v (rest g))]))

;; (find-path orig dest g) finds path from orig to dest in g if it exists
;; find-path: Node Node Graph --> (anyof (listof Node) false)
(define (find-path orig dest g)
  (cond [(symbol=? orig dest) (list dest)]
        [else (local [(define nbrs (neighbours orig g))
                      (define ?path (find-path/list nbrs dest g))]
                (cond [(false? ?path) false]
                      [else (cons orig ?path)])))))

;; (find-path/list nbrs dest g) produces path from
;;   an element of nbrs to dest in g, if one exists
;; find-path/list: (listof Node) Node Graph --> (anyof (listof Node) false)
(define (find-path/list nbrs dest g)
  (cond [(empty? nbrs) false]
        [else (local [(define ?path (find-path (first nbrs) dest g))]
                  (cond [(false? ?path)
                        (find-path/list (rest nbrs) dest g)]
                        [else ?path]))]))
```

Logic paradigm – hướng logic

answer a *question* via searching for a *solution*

- 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

Logic paradigm – hướng logic

Ví dụ đoạn chương trình tìm đường đi trên đồ thị cho bởi danh sách cạnh bằng ngôn ngữ lập trình logic Prolog

```
edge(a,b).  
edge(a,f).  
edge(b,c).  
edge(c,a).  
edge(d,e).  
edge(e,a).  
edge(e,c).  
  
dumb_path(Start,Finish) :- edge(Start,Finish).  
dumb_path(Start,Finish) :- edge(Start,Next),dumb_path(Next,Finish).  
  
path(Start,Finish) :- smart_path(Start,Finish,[]).  
  
smart_path(Current,Target,Visited) :- edge(Current,Target).  
  
smart_path(Current,Target,Visited) :-  
    edge(Current,Next),non_member(Next,Visited),  
    smart_path(Next,Target,[Next|Visited]).  
  
non_member(Elt,[]).  
non_member(Elt,[Hd | Tl]) :- Elt \== Hd, non_member(Elt,Tl).
```

Object-oriented paradigm – hướng đối tượng

send messages between *objects* to simulate a temporal evolution of a set of *real world phenomena*

- Ý tưởng: Các khái niệm và mô hình tương tác trong thế giới thực
- Dữ liệu cũng như các thao tác trên dữ liệu được bao gói trong các đối tượng
- Cơ chế che giấu thông tin nội bộ được sử dụng để tránh những tác động từ bên ngoài

Object-oriented paradigm – hướng đối tượng

- Các đối tượng tương tác với nhau qua việc truyền thông điệp, đó là phép ẩn dụ cho việc thực hiện các thao tác trên 1 đối tượng
- Trong phần lớn các NNLT HĐT, đối tượng phân loại thành các lớp
 - *Đối tượng trong các lớp có chung các thuộc tính, cho phép lập trình trên lớp, thay vì lập trình trên từng đối tượng riêng lẻ*
 - *Lớp đại diện cho các khái niệm còn đối tượng đại diện cho thể hiện*
 - *Lớp có tính kế thừa, cho phép mở rộng hay chuyên biệt hóa*

Giới thiệu về ngôn ngữ C++

Lịch sử ngôn ngữ C

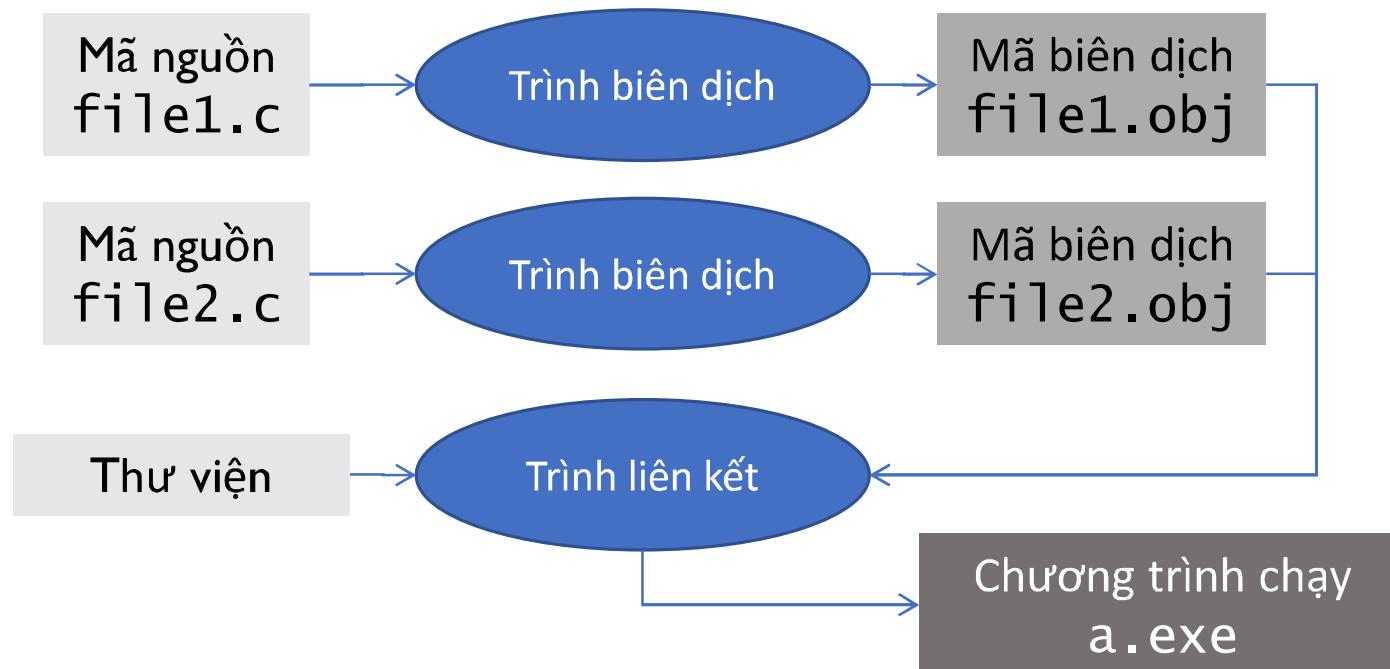
- Ra đời trong những năm 1970, gắn liền với sự phát triển của HĐH Unix. Tác giả: Dennis Ritchie
- Mục tiêu:
 - Đè cao tính hiệu quả
 - Có khả năng truy xuất phần cứng ở cấp thấp
 - Ngôn ngữ có cấu trúc (thay cho lập trình bằng hợp ngữ)
- C là ngôn ngữ trung gian giữa cấp thấp...
 - Có khả năng truy xuất bộ nhớ trực tiếp
 - Cú pháp ngắn gọn, ít từ khoá
- ... và cấp cao
 - Không phụ thuộc phần cứng
 - Cấu trúc, hàm, khả năng đóng gói
 - Kiểm tra kiểu dữ liệu

Lịch sử ngôn ngữ C++

- Ra đời năm 1979 bằng việc mở rộng ngôn ngữ C. Tác giả: Bjarne Stroustrup
- Mục tiêu:
 - Thêm các tính năng mới
 - Khắc phục một số nhược điểm của C
- Bổ sung những tính năng mới so với C:
 - Lập trình hướng đối tượng (OOP)
 - Lập trình tổng quát (template)
 - Nhiều tính năng nhỏ giúp lập trình linh hoạt hơn nữa (thêm kiểu bool, khai báo biến bất kỳ ở đâu, kiểu mạnh, định nghĩa chồng hàm, namespace, xử lý ngoại lệ,...)

Biên dịch chương trình C/C++

- Là quá trình chuyển đổi từ mã nguồn (do người viết) thành chương trình ở dạng mã máy để có thể thực thi được



Biên dịch chương trình C/C++

- Cho phép dịch từng file riêng rẽ giúp:
 - Dễ phân chia và quản lý từng phần của chương trình
 - Khi cần thay đổi, chỉ cần sửa đổi file liên quan
 - giảm thời gian bảo trì, sửa đổi
 - Chỉ cần dịch lại những file có thay đổi khi cần thiết
 - giảm thời gian dịch
- Các trình biên dịch hiện đại còn cho phép tối ưu hóa dữ liệu và mã lệnh
- Một số trình biên dịch thông dụng: MS Visual C++, gcc, Intel C++ Compiler, Watcom C/C++,...

Vào ra trong C++

Header file cho I/O trong C++

Header File	Miêu tả
<iostream>	File này định nghĩa các đối tượng cin, cout, cerr và clog, tương ứng với Standard Input Stream (Luồng đầu vào chuẩn), Standard Output Stream (Luồng đầu ra chuẩn), Un-buffered Standard Error Stream (Luồng lỗi chuẩn không được đệm) và Buffered Standard Error Stream (Luồng lỗi chuẩn được đệm).
<iomanip>	File này khai báo các dịch vụ hữu ích để thực hiện hoạt động I/O được định dạng với các bộ thao tác luồng được tham số hóa như setw và setprecision.
<fstream>	File này khai báo các dịch vụ xử lý file được kiểm soát bởi người dùng. Chúng ta sẽ thảo luận chi tiết về nó trong chương File và Stream trong C++

Standard Output Stream (cout) trong C++

- Đối tượng tiền định nghĩa cout là một minh họa của lớp ostream. Đối tượng cout được xem như "được kết nối tới" thiết bị đầu ra chuẩn, thường là màn hình. Đối tượng cout được sử dụng kết hợp với toán tử chèn luồng (insertion operator), được viết là <<, như ví dụ dưới đây:

```
#include <iostream>
using namespace std;
int main( ) {
    char str[] = "Xin chao C++";
    cout << "Gia tri cua str la: " << str << endl;
}
```

- Toán tử chèn luồng << có thể được sử dụng nhiều hơn một lần trong một lệnh và endl được sử dụng để thêm một dòng mới tại cuối dòng đó.

Standard Input Stream (cin) trong C++

- Đối tượng tiền định nghĩa **cin** là một minh họa của lớp **isrstream**. Đối tượng **cin** được xem như đính kèm với thiết bị đầu vào chuẩn, mà thường là bàn phím. Đối tượng **cin** được sử dụng kết hợp với toán tử trích luồng (extraction operator), viết là **>>**, như trong ví dụ sau:

```
#include <iostream>
using namespace std;
int main( ) {
    char ten[50];
    cout << "Nhap ten cua ban (viet lien): ";
    cin >> ten;
    cout << "Ten ban la: " << ten << endl;
}
```

Standard Input Stream (cin) trong C++

- Bộ biên dịch C++ cũng quyết định kiểu dữ liệu của giá trị đã nhập và chọn toán tử trích luồng thích hợp để trích giá trị và lưu giữ nó trong các biến đã cung cấp.
- Toán tử trích luồng >> có thể được sử dụng nhiều hơn một lần trong một lệnh. Để yêu cầu nhiều hơn một dữ liệu chuẩn, bạn có thể sử dụng:

```
cin >> ten >> tuoi;
```

Nó tương đương với hai lệnh sau:

```
cin >> ten; cin >> tuoi;
```

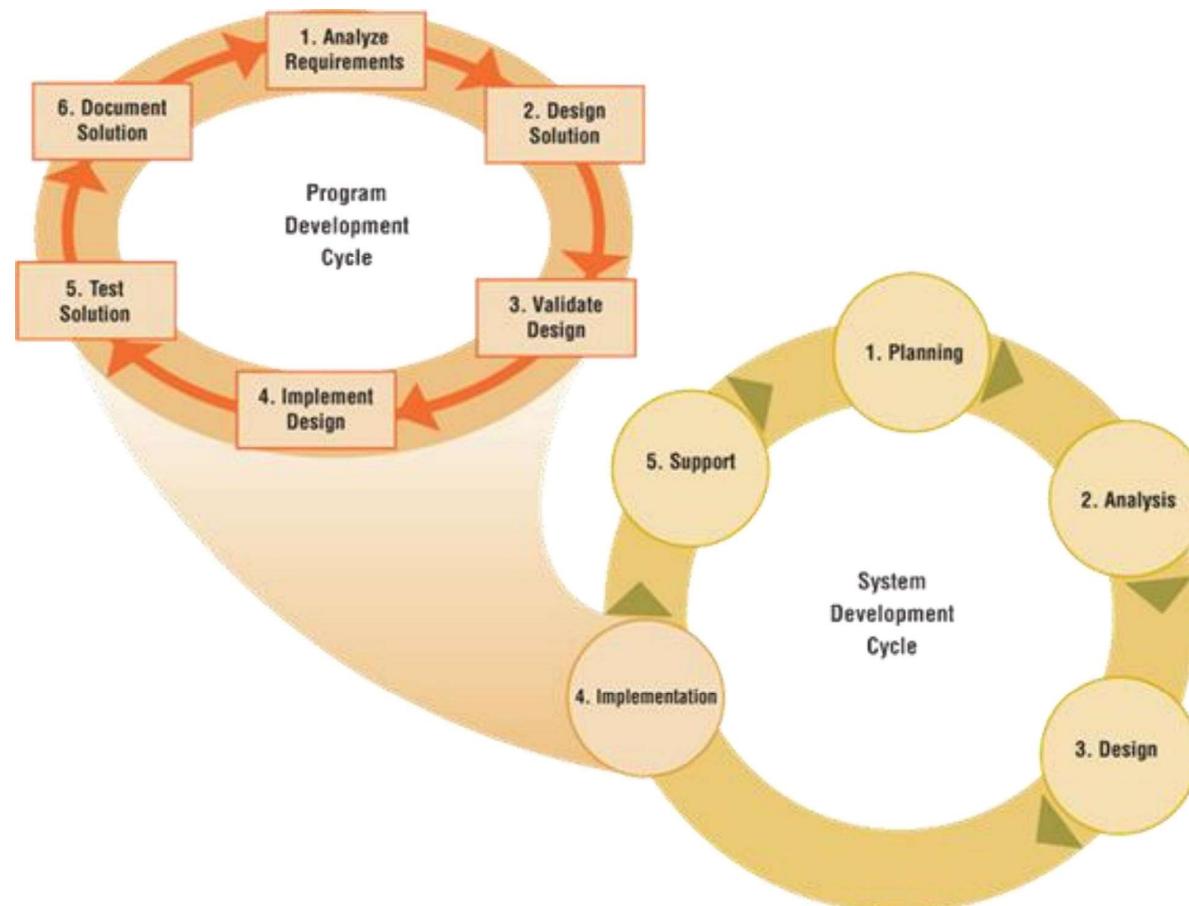
IDE lập trình

- Codeblock: <http://www.codeblocks.org/downloads/26>
- Link download: <https://www.fosshub.com/Code-Blocks.html?dwI=codeblocks-17.12mingw-setup.exe>

Chu trình phát triển phần mềm

Chu trình phát triển phần mềm

- Program development cycle?
- Là các bước mà lập trình viên dùng để xây dựng chương trình



Step 1 – Analyze Requirements

- Các việc cần làm khi phân tích yêu cầu?
 1. Thiết lập các requirements
 2. Gặp các nhà phân tích hệ thống và users
 3. Xác định input, output, processing, và các thành phần dữ liệu

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.	

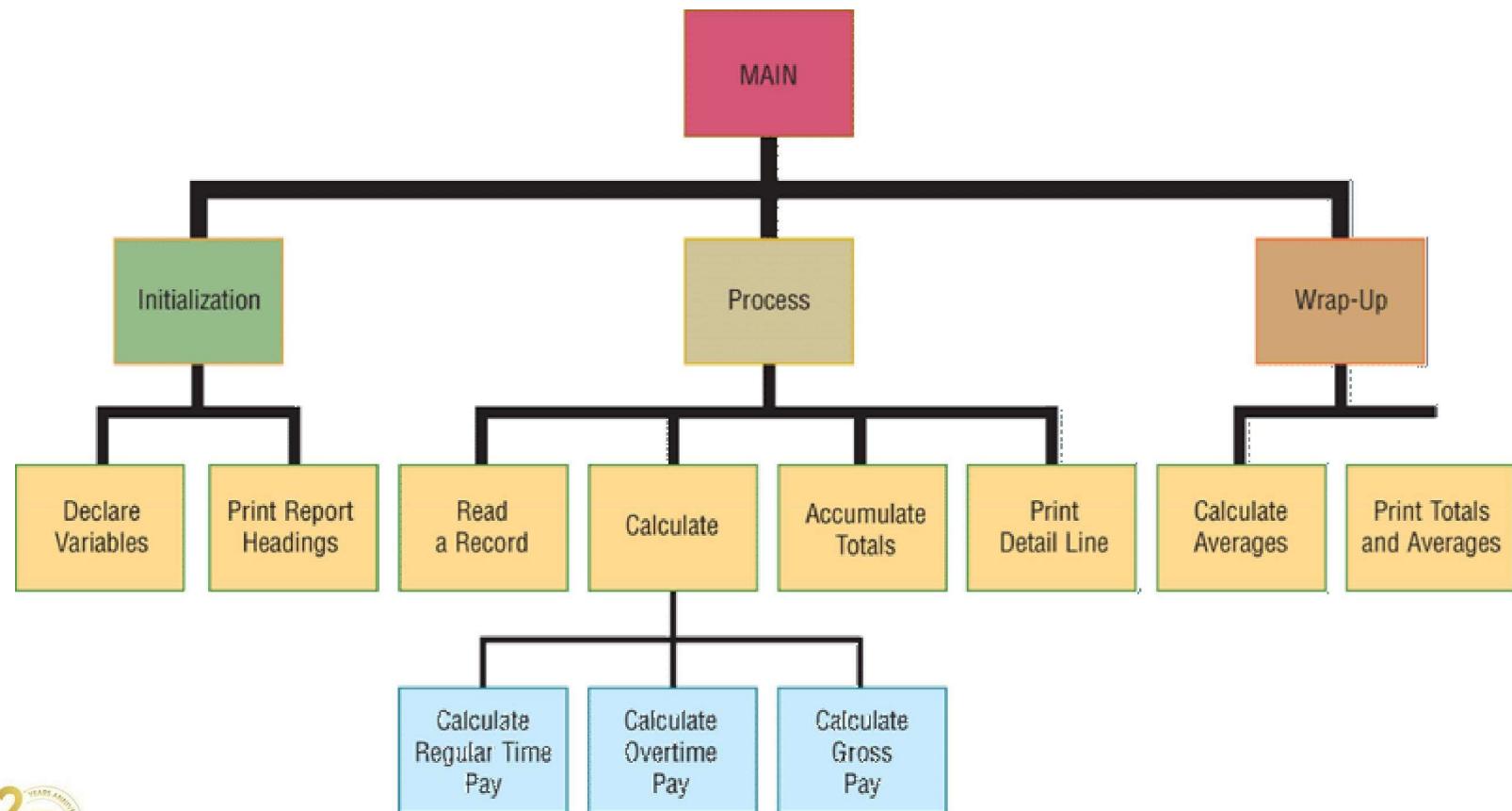
Step 2 – Design Solution

- Những việc cần làm trong bước thiết kế giải pháp?



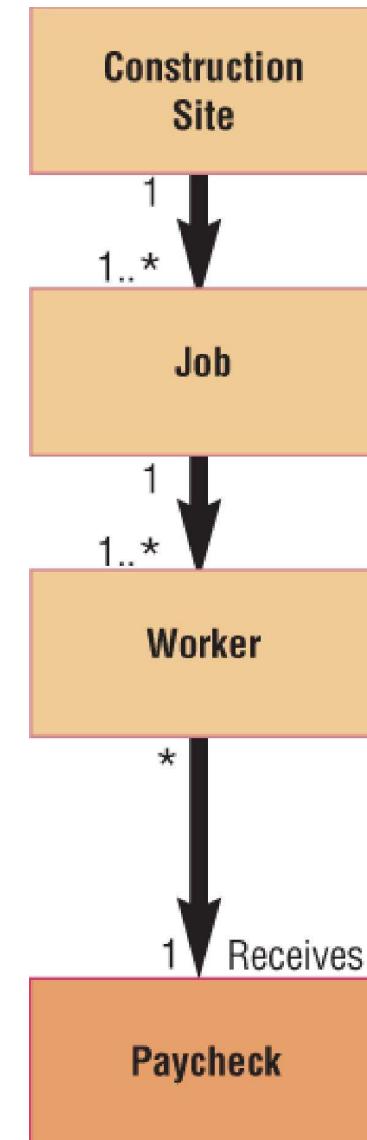
Step 2 – Design Solution

- Sơ đồ phân cấp chức năng- hierarchy chart?
- Trực quan hóa các modules chương trình
- Còn gọi là sơ đồ cấu trúc



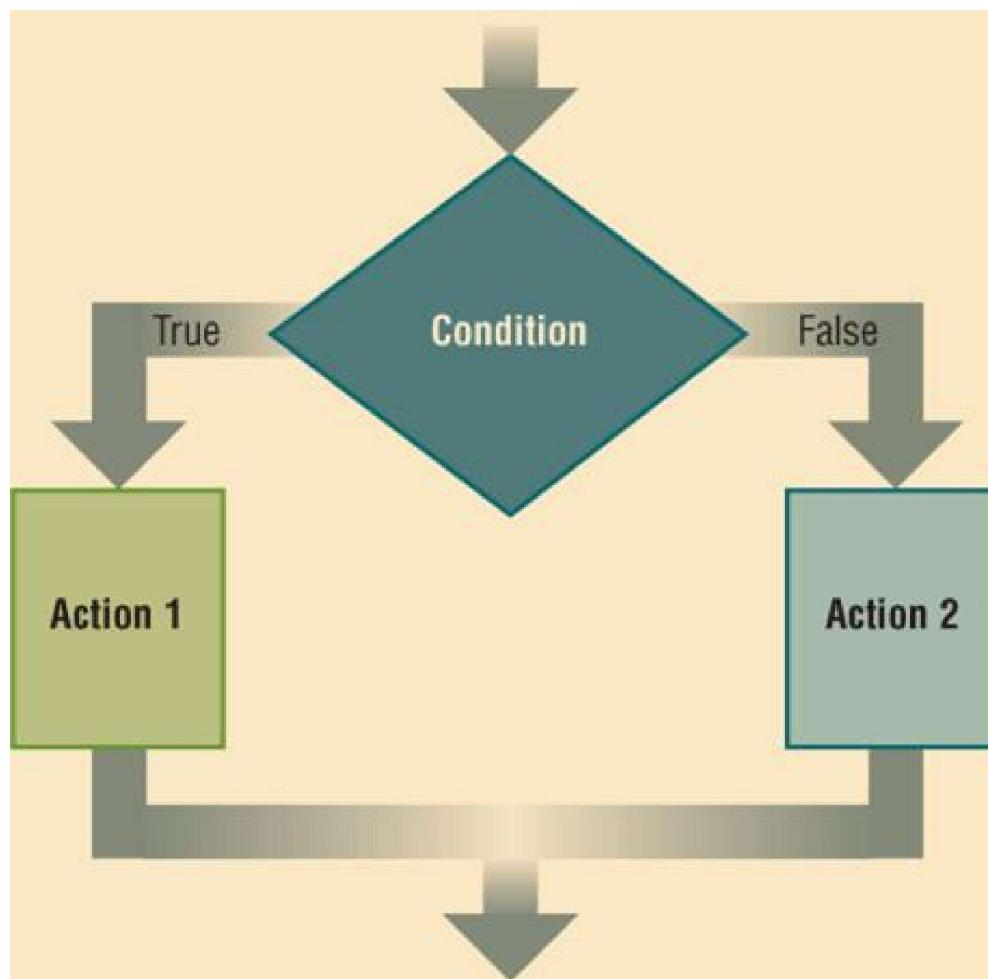
Step 2 – Design Solution

- 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

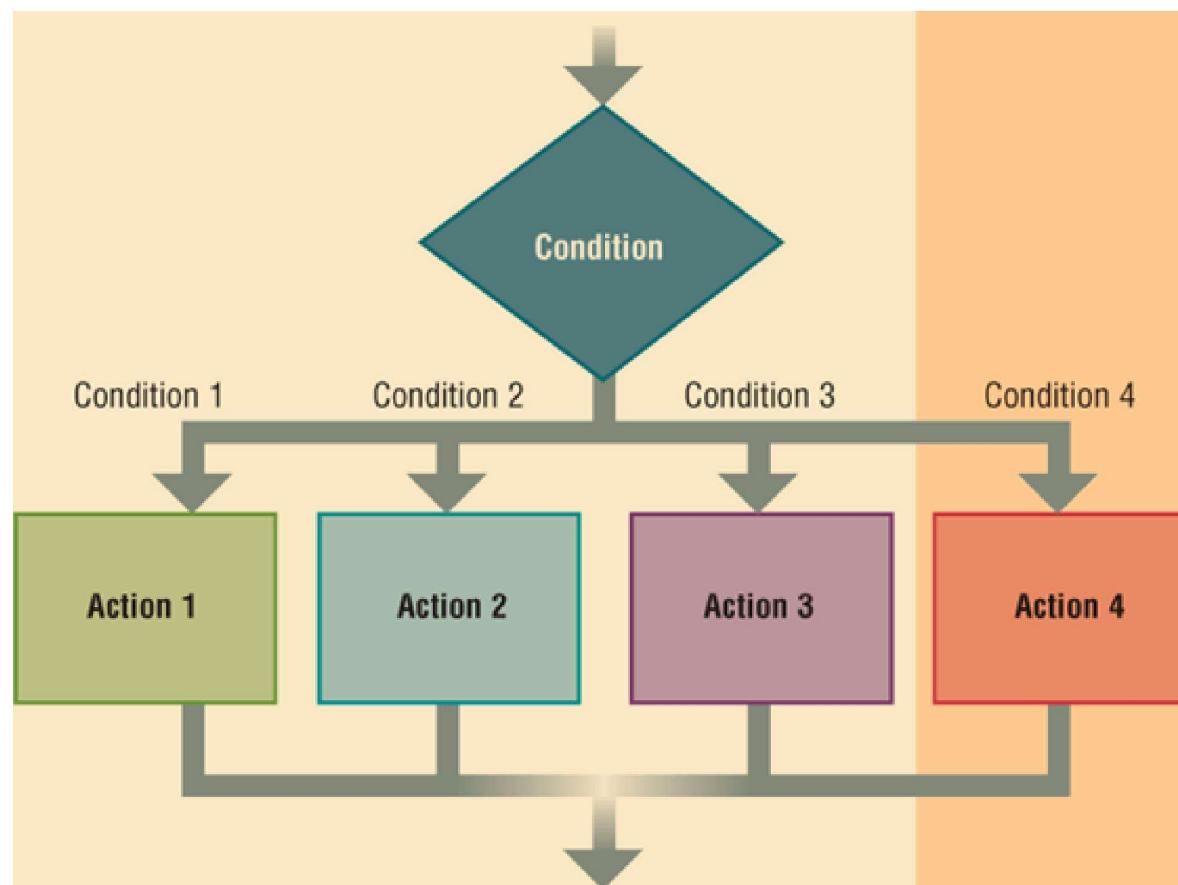
- Cấu trúc rẽ nhánh



- Chỉ ra action tương ứng điều kiện
- 2 kiểu
 - Case control structure
 - If-then-else control structure—dựa theo 2 khả năng: true or false

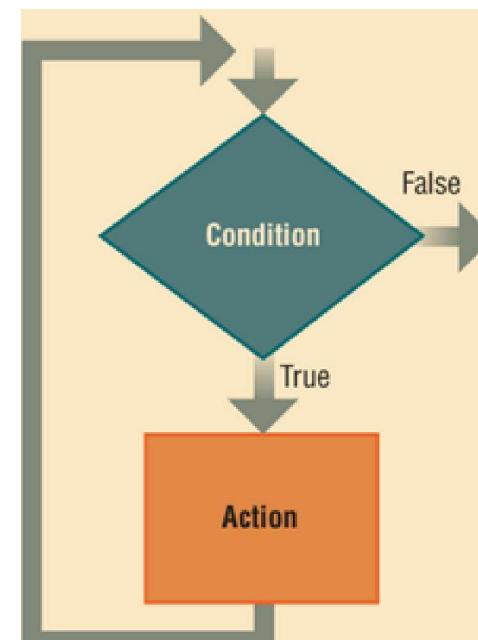
Step 2 – Design Solution

- Case control structure
- Dựa theo 3 hoặc nhiều hơn các khả năng

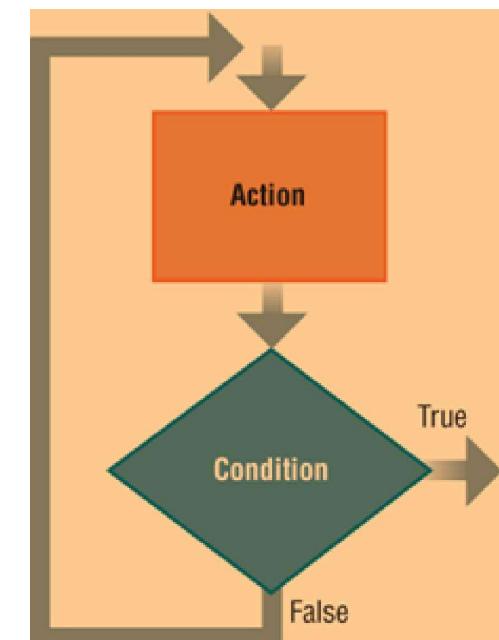


Step 2 – Design Solution

- Cấu trúc lặp
- Cho phép CT thực hiện 1 hay nhiều actions 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



Do-While Control Structure



Do-Until Control Structure

Step 3 – Validate Design

- Những điều cần làm trong giai đoạn này?

Kiểm tra
độ chính xác
của chương trình

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

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

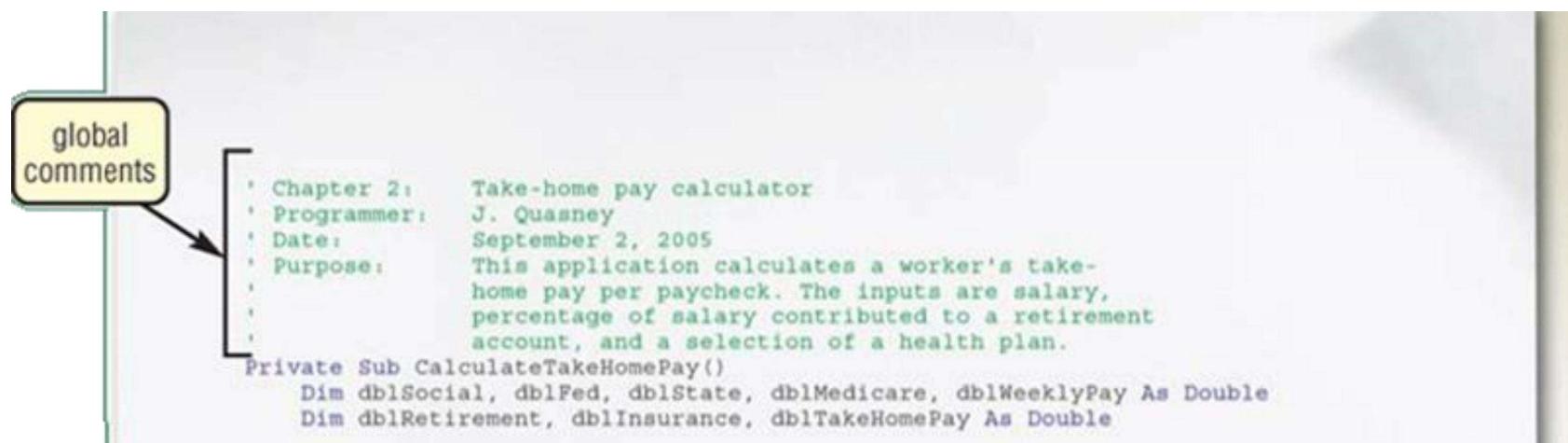
LTV kiểm tra
logic cho tính đúng đắn
và thử tìm các lỗi logic

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

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 kq 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

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

Rà soát, hoàn thiện documentation



25 YEARS ANNIVERSARY
SOICT

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Xin cảm ơn!

