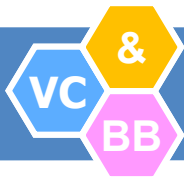


KỸ THUẬT LẬP TRÌNH



1. Tổng quan về Ngôn ngữ lập trình và thiết kế chương trình



Nội dung

1

Ngôn ngữ lập trình

2

Chu trình phát triển chương trình

3

Nhắc lại ngôn ngữ C/C++

4

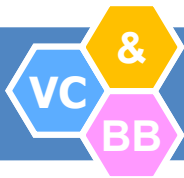
Trắc nghiệm

❖ Computer program?

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



- ❖ *Một NNLT là một hệ thống các ký hiệu dùng để liên lạc, trao đổi một nhiệm vụ/ thuật toán với máy tính, làm cho nhiệm vụ được thực thi. Nhiệm vụ được thực thi gọi là một *computation*, nó 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 hình ngôn ngữ-Language paradigm* là những nguyên tắc chung cơ bản, dùng bởi LTV để xd chương trình.
 - *Cú pháp - Syntax* 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 – semantics* của chương trình trong ngôn ngữ ấy.
- ❖ 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 một tổ chức để phục vụ cho bản thân họ.

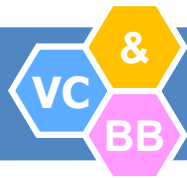


Cont...

- ❖ Về cơ bản, chỉ có 4 mô hình NNLT 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



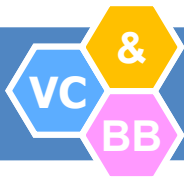
- ❖ Computer program được nạp vào *primary memory* như là 1 tập các lệnh bằng ngôn ngữ máy, tức là một dãy tuần tự các số nhị phân - *binary digits*.
- ❖ Tại bất cứ một thời điểm nào, computer sẽ ở một trạng thái *state* nào đó.
- ❖ Đặc điểm cơ bản của trạng thái là con trỏ lệnh *instruction pointer* trỏ tới lệnh mã máy 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 program 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
```

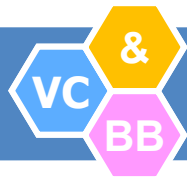


ASSEMBLY LANGUAGE

- ❖ 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

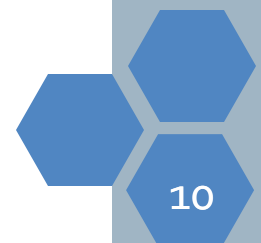
```
movl    #0x1,n
compare:
    cmpl    #0xa,n
    cgt     end_of_loop
    acddl   #0x1,n
    bra     compare
end_of_loop:
```

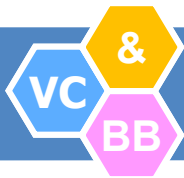

- ❖ Thay vì dựa trên phần cứng (machine-oriented) cần tìm cơ chế dựa trên vấn đề (problem-oriented) để tạo chương trình.
- ❖ Chính vì thế *high(er) level* languages – là các ngôn ngữ lập trình gần với ngôn ngữ con người hơn – dùng các từ khóa giống tiếng anh – đã được xây dựng như : Algol, Fortran, Pascal,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



PHÂN LOẠI THEO THỜI GIAN

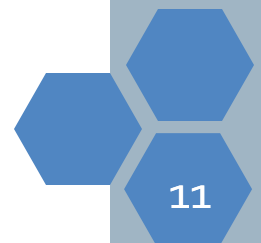
- ❖ 1940s : Machine code
- ❖ 1950s Khai thác sức mạnh của MT: Assembler code, Autocodes, first version of Fortran
- ❖ 1960s Tăng khả năng tính toán: Cobol, Lisp, Algol 60, Basic, PL/1 --- nhưng vẫn dùng phong cách lập trình cơ bản của assembly language.
- ❖ 1970s Bắt đầu cuộc khủng hoảng phần mềm “software 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 xd.
- ❖ 2000s Genetic programming languages, DNA computing, bio-computing?
- ❖ Trong tương lai : Ngôn ngữ It lượng tử : Quantum ?



- ❖ Khái niệm software crisis bao gồm hàng loạt vấn đề nảy sinh trong việc phát triển phần mềm trong những năm 1960s khi muốn xd 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. Chất lượng phần mềm bị giảm, thấp.
- ❖ Để giải quyết các vấn đề kể trên , chuyên ngành software engineering ra đời.

❖ Low-level languages và high-level languages?

Low-level language

Machine-dependent

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

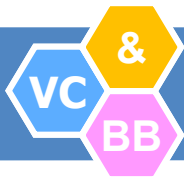
Machine và assembly languages là ngôn ngữ bậc thấp low-level

High-level language

Machine-independent

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

❖ Level	Instructions	Memory handling
Low level languages	Dạng bits – giống các lệnh machine	Truy cập và cấp phát trực tiếp bộ nhớ
High level languages	Dùng các biểu thức và các dòng điều khiển xác định	Truy cập và cấp phát bộ nhớ qua các lệnh, toán tử - operators
Very high level languages	Hoàn toàn trừu 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 : “Cái gì cần lưu trữ” và “Lưu trữ như thế nào”.
- ❖ 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 mệnh lệnh).

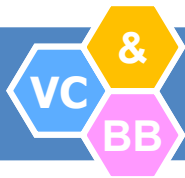
❖ Procedural language?

Lập trình viên viết các chỉ thị hướng dẫn cho máy tính cái gì cần làm và làm như thế nào

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

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

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

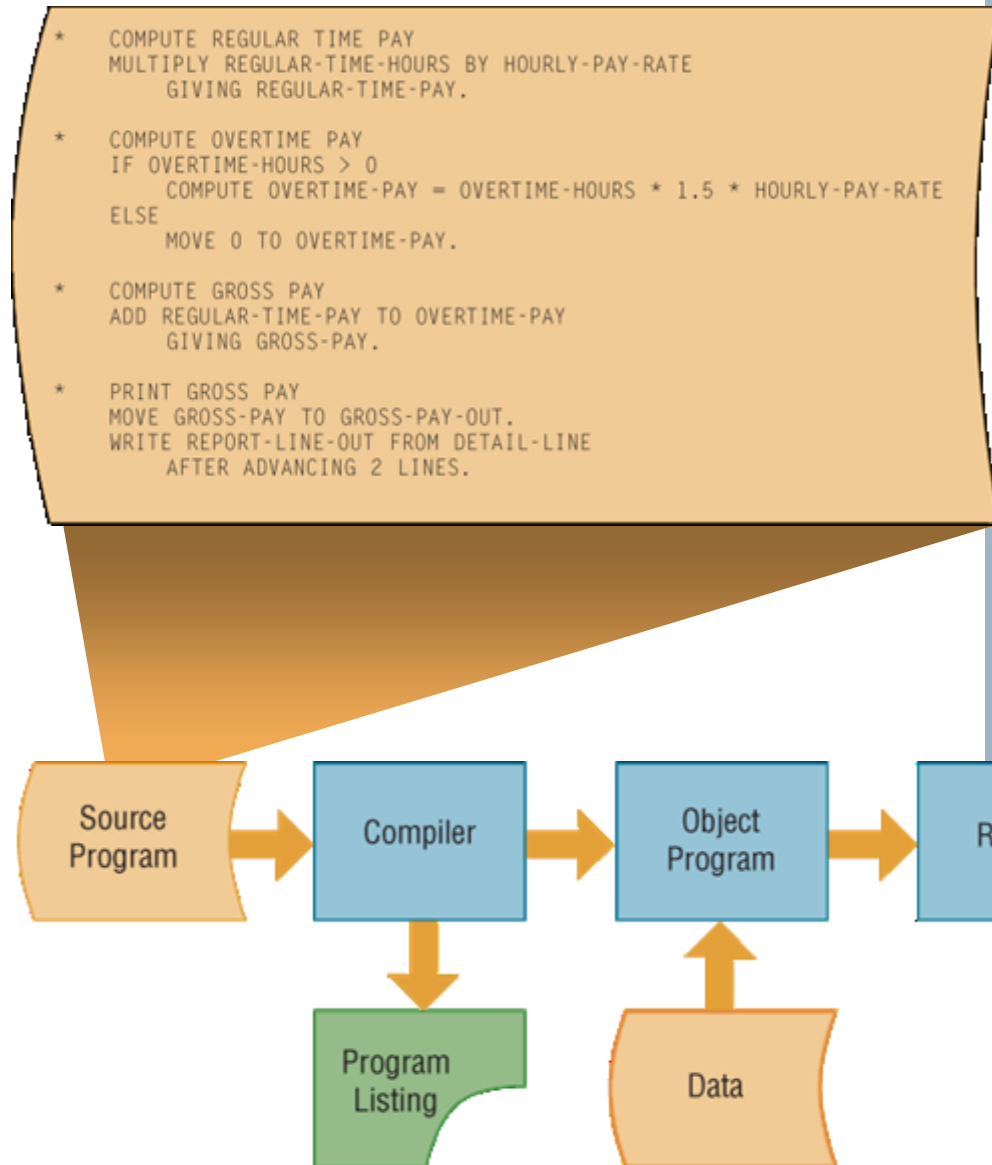


Procedural Languages

❖ Trình dịch –

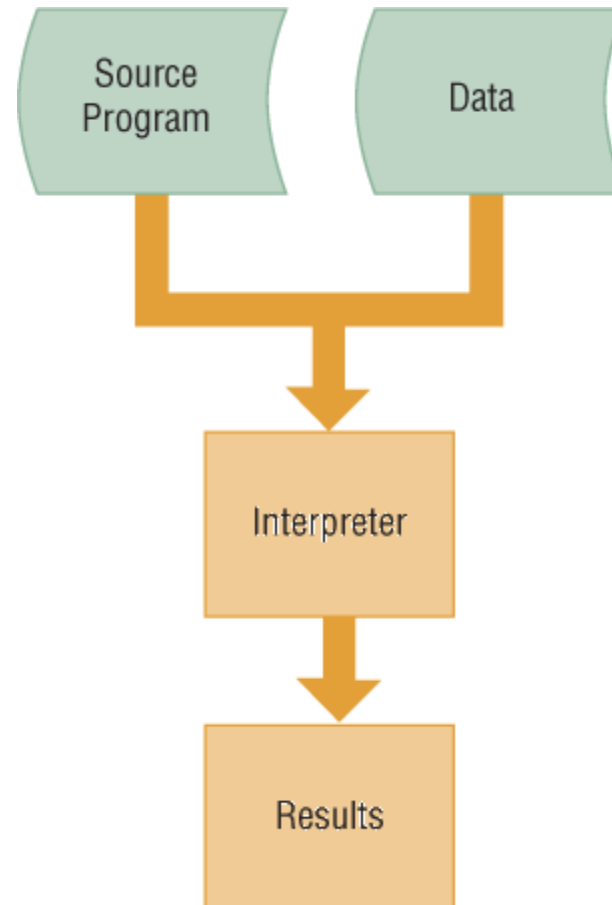
Compiler?

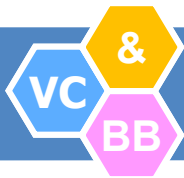
- Là chương trình thực hiện biên dịch toàn bộ chương trình nguồn thành mã máy trước khi thực hiện



❖ Thông dịch - Interpreter?

- Là chương trình dịch và thực hiện từng dòng lệnh của chương trình cùng lúc
- Không tạo ra object program





Procedural Languages

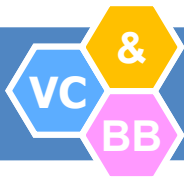
❖ BASIC?

- Được thiết kế để cho những người mới học tiếp can 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
- **CO**mmon **B**usiness-**O**riented **L**anguage

```
* 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.
```

❖ C?

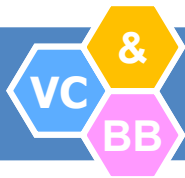
- Là NNLT rất mạnh, ban đầu được thiết kế để lập trình hệ thống -write system software
- Yêu cầu những kỹ năng lập trình chuyên nghiệp

```
/* 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**

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

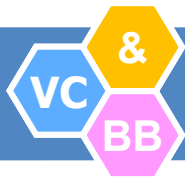
**Lợi ích cơ bản là
khả năng tái sử
dụng -reuse
existing objects**

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

Event là hành
động mà
chương trình
cần đáp ứng

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





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**

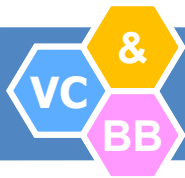
```
// portion of a C++ program that allows users to create a new zip code from a
// string or a number and expand zip codes, as appropriate, to a 10-digit number

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);
    }
}
```



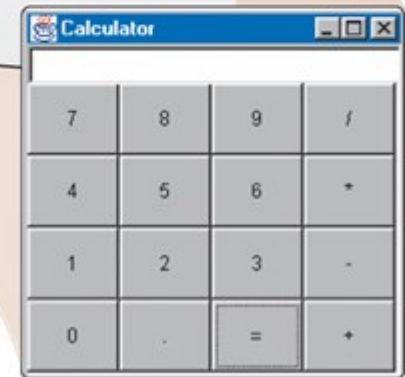
Object-Oriented Programming Languages

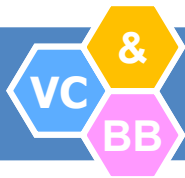
❖ Java?

- 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?

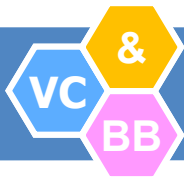
Visual programming environment (VPE)
Cho phép developers kéo và thả các objects để xd programs

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



Nội dung

1

Ngôn ngữ lập trình

2

Chu trình phát triển chương trình

3

Nhắc lại ngôn ngữ C/C++

4

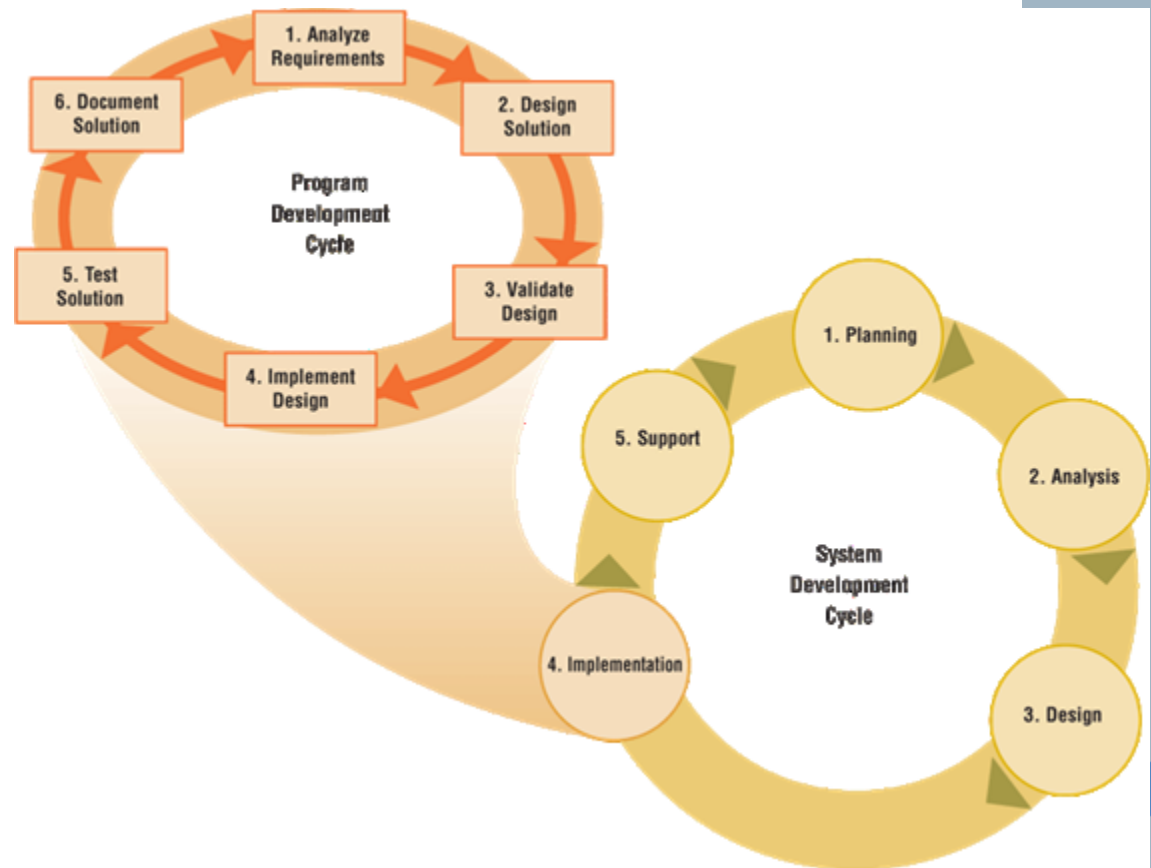
Trắc nghiệm

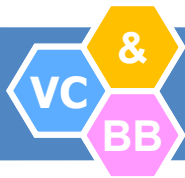
Chu trình phát triển Program

❖ Program development cycle?

➤ Là các bước mà LTV dùng để xây dựng programs

- **Programming team**—Nhóm LTV cùng xd 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**

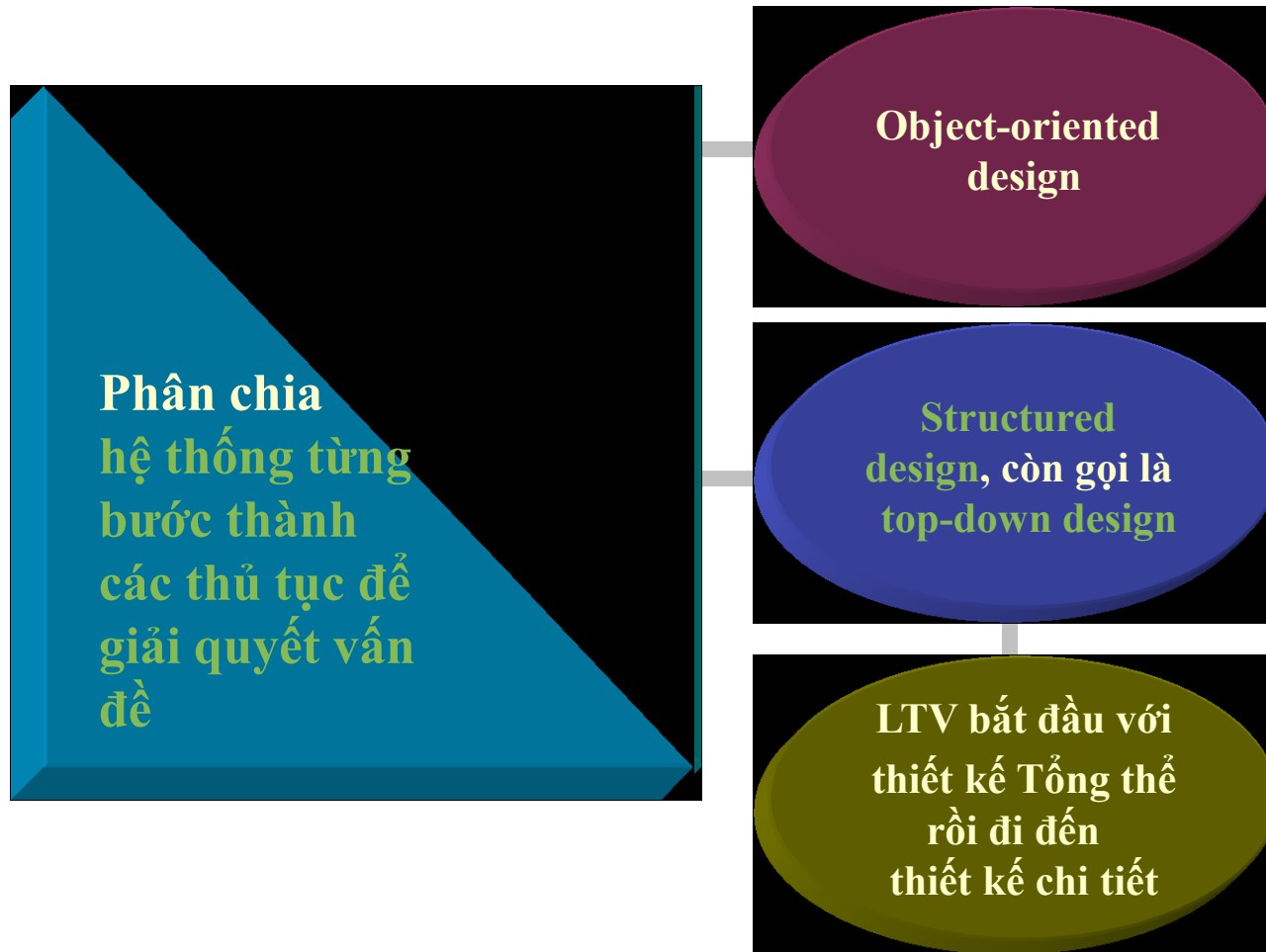
- **IPO chart**—Xác định đầu vào, đầu ra và các bước xử lý

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

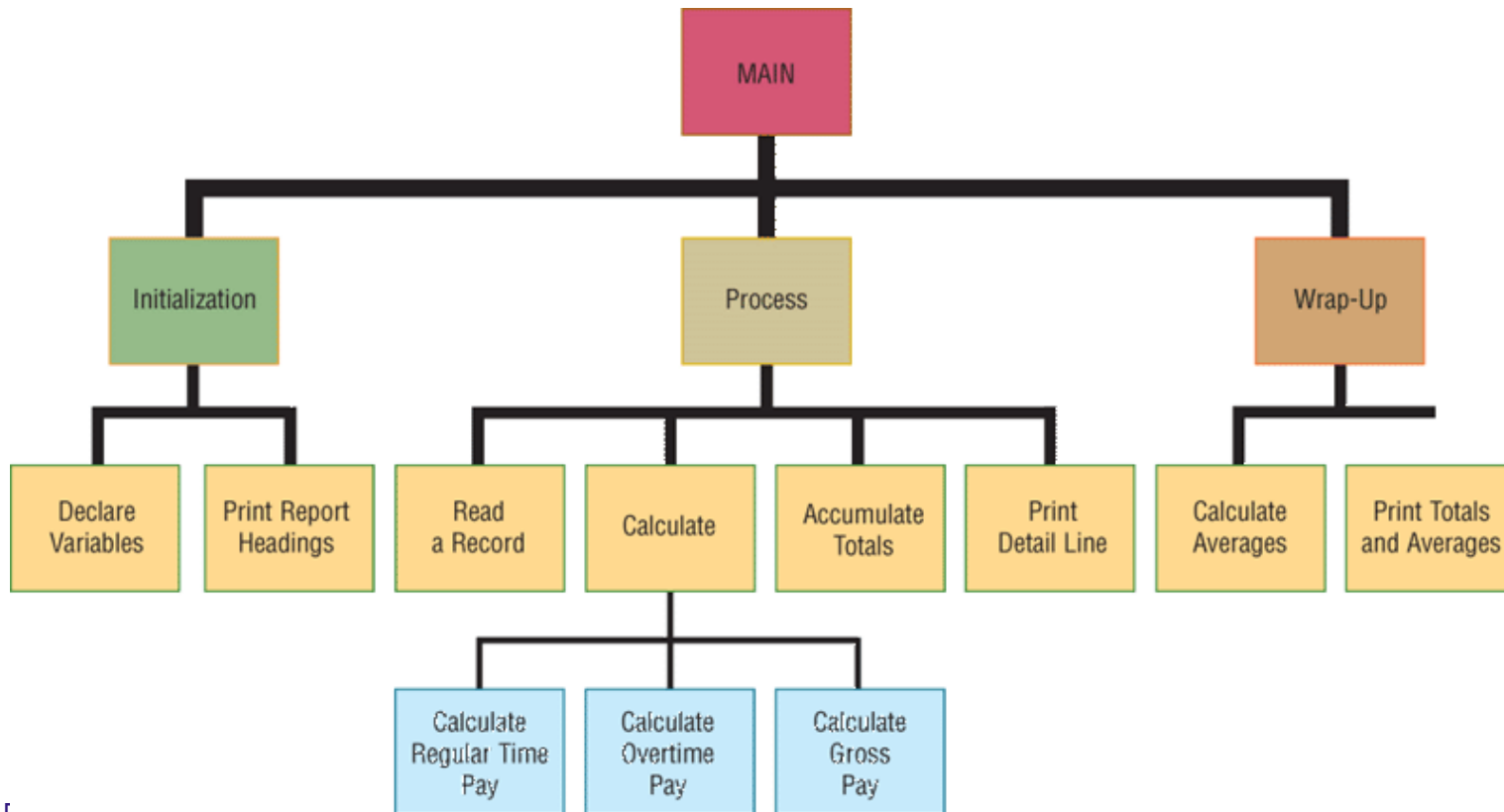
❖ Những việc cần làm trong bước thiết kế giải pháp?



Step 2 — Design Solution

❖ Sơ đồ phân cấp chức năng- hierarchy chart?

- **Trực quan hóa các modules ct**
- **Còn gọi là sơ đồ cấu trúc**

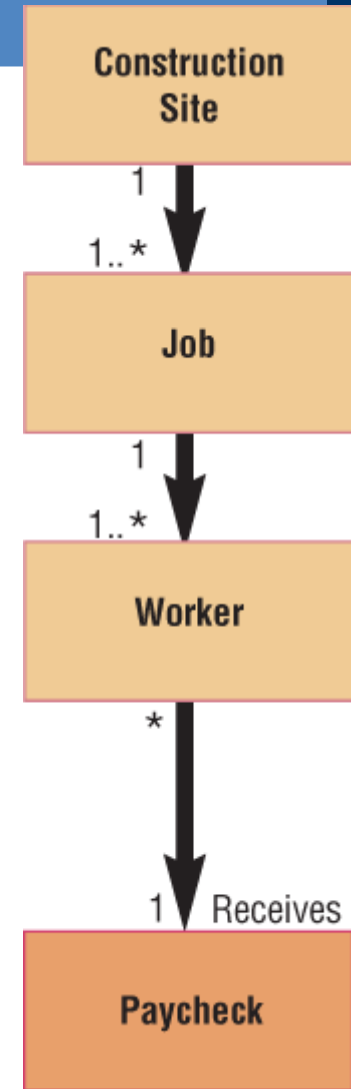


Step 2 — Design Solution

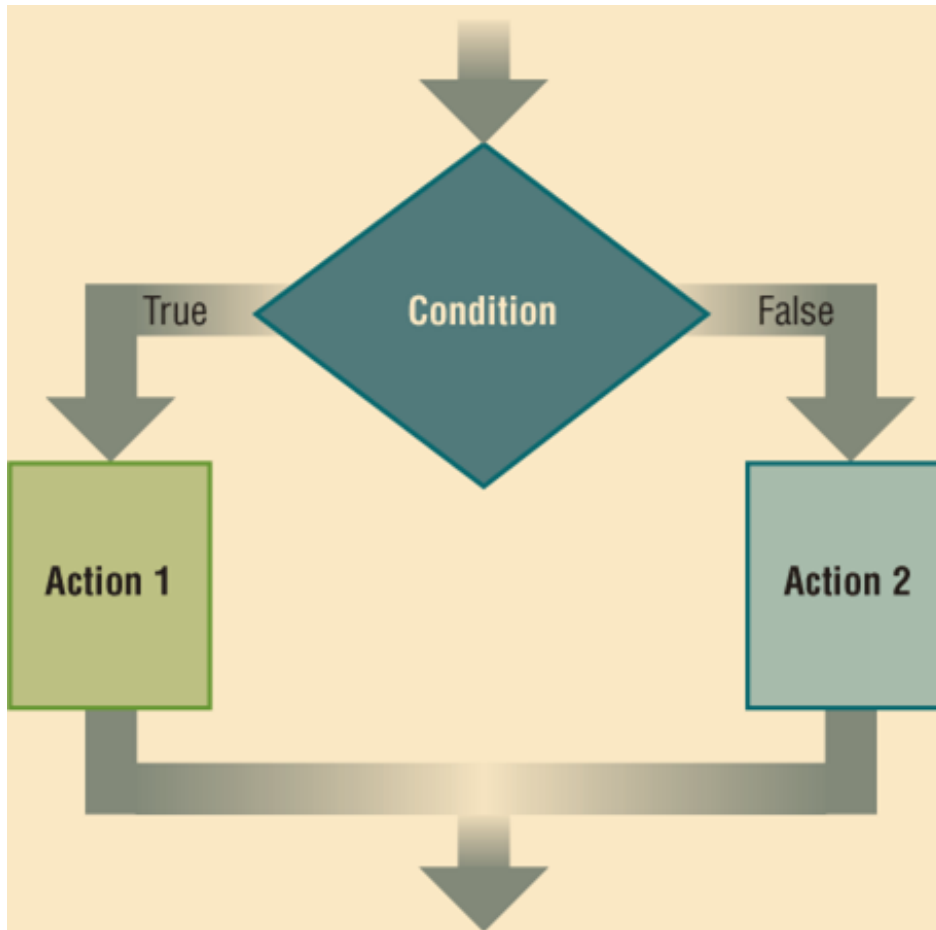
❖ Object-oriented (OO) design là gì?

➤ 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



❖ Cấu trúc tuyến chọn



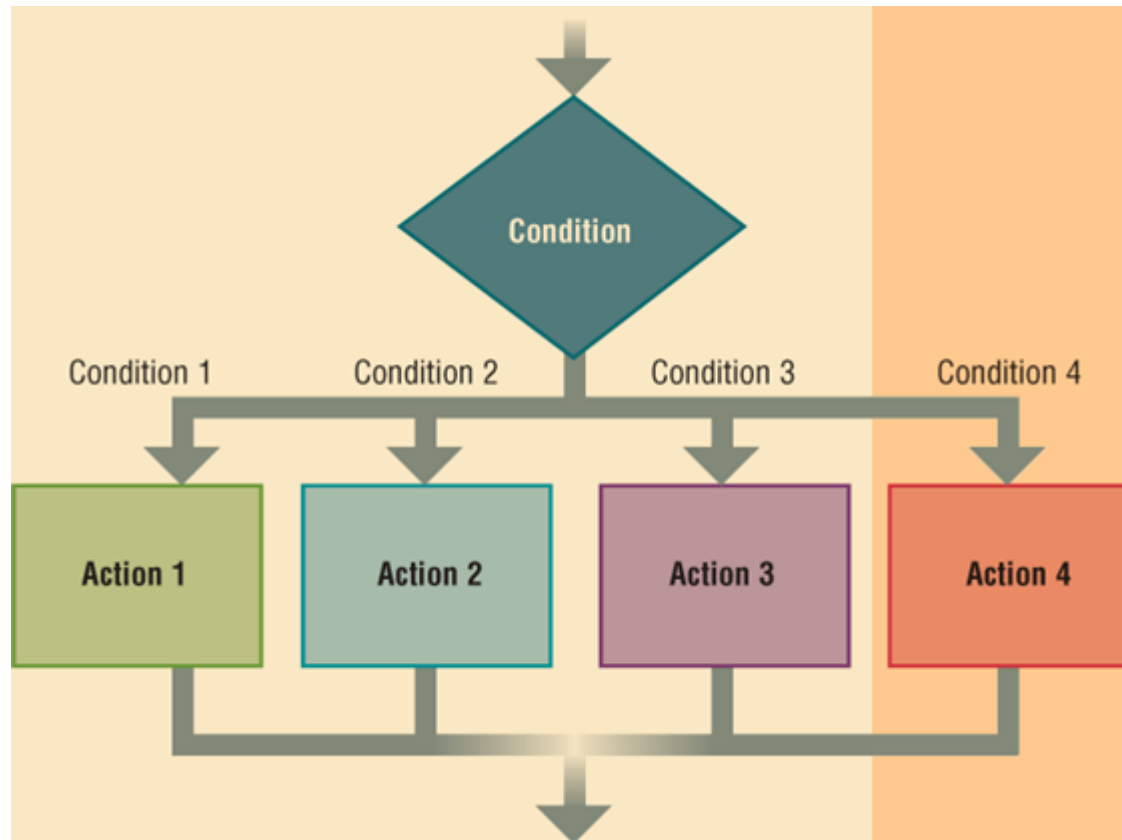
- **Chỉ ra action tương ứng điều kiện**
- **2 kiểu**

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

Step 2 — Design Solution

❖ Case control structure

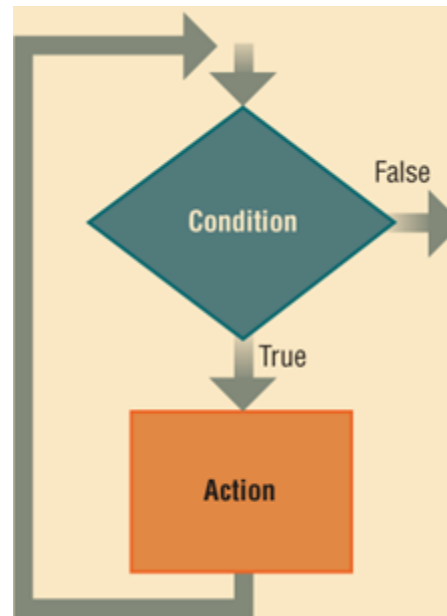
- **Dựa theo 3 hoặc nhiều hơn các khả năng**



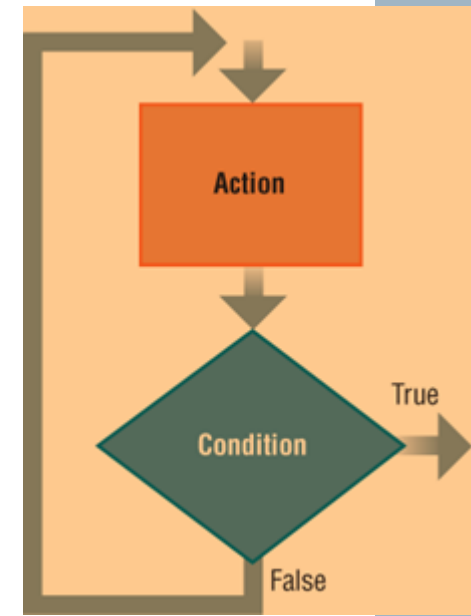
❖ Cấu trúc lặp

➤ Cho phép CT thực hiện 1 hay nhiều actions lặp đi lặp lại

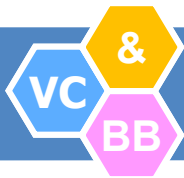
- **Do-while control structure**—lặp khi điều kiện còn đúng
- **Do-until control structure**—Lặp cho đến khi điều kiện đúng



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 program**

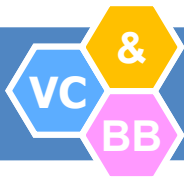
Desk check
LTV dùng các dữ liệu
thử nghiệm để kiểm tra ct

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

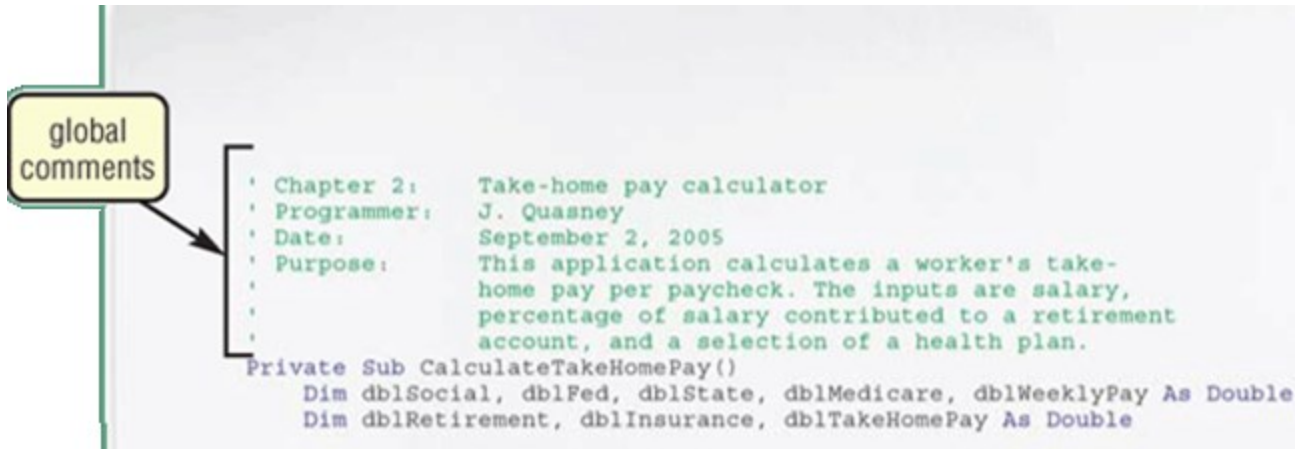
Structured walkthrough
LTV mô tả logic
của thuật toán trong khi
programming team duyệt theo
logic chương trình



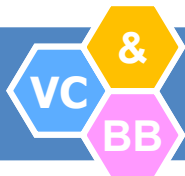
Step 4 — Implement Design

❖ 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



```
' Chapter 2:      Take-home pay calculator
' Programmer:    J. Quasney
' Date:          September 2, 2005
' Purpose:       This application calculates a worker's take-
'               home pay per paycheck. The inputs are salary,
'               percentage of salary contributed to a retirement
'               account, and a selection of a health plan.
Private Sub CalculateTakeHomePay()
    Dim dblSocial, dblFed, dblState, dblMedicare, dblWeeklyPay As Double
    Dim dblRetirement, dblInsurance, dblTakeHomePay As Double
```



Step 5 — Test Solution

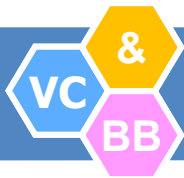
❖ Những việc cần làm ?

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

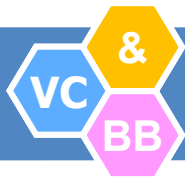
❖ Là bước không kém quan trọng

➤ **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



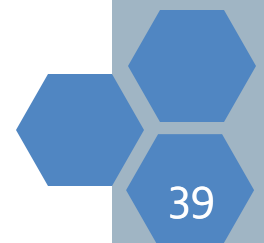


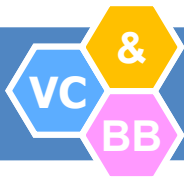
Tóm lại

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 hình lập trình cơ bản





Nội dung

1

Ngôn ngữ lập trình

2

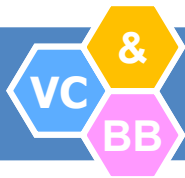
Chu trình phát triển chương trình

3

Nhắc lại ngôn ngữ C/C++

4

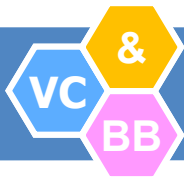
Trắc nghiệm



Lịch sử của ngôn ngữ C/C++

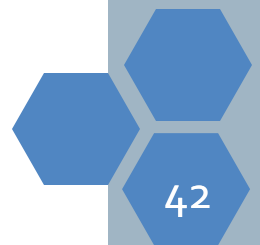
- ❑ C được tạo bởi Dennis Ritchie ở Bell Telephone Laboratories vào năm 1972.
- ❑ Vào năm 1983, học viện chuẩn quốc gia Mỹ (American National Standards Institute - ANSI) thành lập một tiểu ban để chuẩn hóa C được biết đến như ANSI Standard C
- ❑ C++ được xây dựng trên nền tảng ANSI Standard C
- ❑ C++ là một ngôn ngữ lập trình hướng đối tượng, nó bao hàm cả ngôn ngữ C

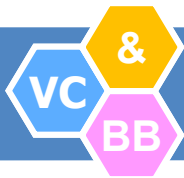




Khảo sát một chương trình C/C++ đơn giản

```
// my first program in C/C++  
#include <conio.h>  
#include <iostream.h>  
int main()  
{  
    cout << "Hello World!"; //Output "Hello World!"  
    getch();  
    return 0;  
}
```





Khảo sát một chương trình C/C++ đơn giản

// my first program in C/C++ :

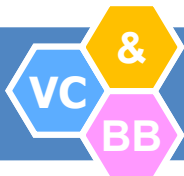
dòng chú thích, không ảnh hưởng đến hoạt động của chương trình

#include <iostream.h>:

Các lệnh bắt đầu bằng dấu # gọi là chỉ thị tiền xử lý (preprocessor)

int main():

- Hàm main là điểm mà tất cả các chương trình C/C++ bắt đầu thực hiện.
- Hàm main không phụ thuộc vào vị trí của hàm
- Nội dung trong hàm main luôn được thực hiện đầu tiên khi chương trình được thực thi
- Chương trình C/C++ phải tồn tại hàm main()
- Nội dung của hàm main() tiếp sau phần khai báo chính thức đặt trong cặp dấu { }



Khảo sát một chương trình C/C++ đơn giản

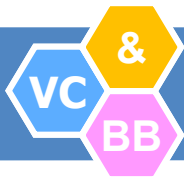
- *cout << "Hello World!";*

Đây là một lệnh nằm trong phần thân của hàm main

- *Cout*: là một dòng (stream) xuất chuẩn C/C++ được định nghĩa trong thư viện `iostream.h`. Khi dòng lệnh thực thi thì dòng lệnh `Hello Word!` được xuất ra màn hình
- *getch()*: dùng để chờ nhập một ký tự từ bàn phím.
- *return 0*: lệnh kết thúc hàm main trả về mã đi sau nó.

Các chú thích

- ❑ Các chú thích được các lập trình viên sử dụng để ghi chú hay mô tả trong các phần của chương trình.
- ❑ Trong C/C++ có hai cách để chú thích:
- ❑ Chú thích dòng: dùng cặp dấu `//`.
- ❑ Chú thích khối (chú thích trên nhiều dòng) dùng cặp `/* ... */`.



Các chú thích

```
/* My second program in C/C++ with more  
comments
```

```
Author: Novice programmer
```

```
Date: 01/01/2008
```

```
*/
```

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

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

```
int main()
```

```
{
```

```
    cout << "Hello World! "; // output Hello World!
```

```
    cout << "I hate C/C++."; // output I hate C/C++.
```

```
    getch();
```

```
    return 0;
```

```
}
```



Cấu trúc của một chương trình C/C++

- ❑ Cấu trúc một chương trình C/C++ gồm: các tiền xử lý, khai báo biến toàn cục, hàm main...

```
/* fact.c  
Purpose: prints the factorials of  
the numbers from 0 through 10  
Author: Mr.Beginner  
Date: 01/01/2008  
*/
```

Phần này thường dùng để mô tả mục đích chương trình, tác giả, ngày viết, ... (Phần không bắt buộc)

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

Khai báo các tập tin thư viện

```
int factorial(int n);
```

Khai báo prototype của các hàm tự tạo

Cấu trúc của một chương trình C/C++

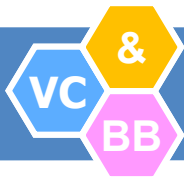
```
int main()
{
    int i;
    for(i=0; i<=10; i++)
        cout<<i<<"!="<<factorial(i);
    return 0;
}
```

Hàm chính của chương trình

```
/* This function computes the
factorial of its parameter, returning it */
```

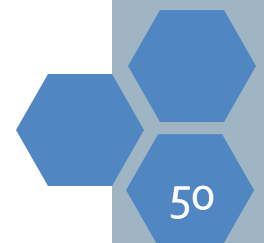
```
int factorial(int n)
{
    int i, product;
    product = 1;
    for (i=2; i<=n; i++) prod *= i;
    return product;
}
```

*Định nghĩa các hàm do người
dùng tự xây dựng*

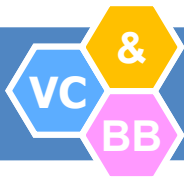


Các tập tin thư viện thông dụng

- ❑ Đây là các tập tin chứa định nghĩa các hàm thông dụng khi lập trình C/C++.
- ❑ Muốn sử dụng các hàm trong các tập tin header này thì phải khai báo `#include <FileName.h>` ở phần đầu của chương trình, với `FileName.h` là tên tập tin thư viện.



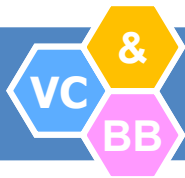
- ❑ Các tập tin thư viện thông dụng gồm:
 - ***Stdio.h(C), iostream.h(C++)***: định nghĩa các hàm vào ra chuẩn như các hàm xuất dữ liệu (printf())/cout), nhập giá trị cho biến (scanf())/cin), nhận ký tự từ bàn phím (getc()), in ký tự ra màn hình (putc()), nhập một chuỗi ký tự từ bàn phím (gets()), xuất chuỗi ký tự ra màn hình (puts())
 - ***Conio.h***: định nghĩa các hàm vào ra trong chế độ DOS, như clrscr(), getch(), ...



Các tập tin thư viện thông dụng

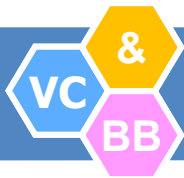
- ***math.h***: Định nghĩa các hàm toán học như: `abs()`, `sqrt()`, `log()`, `log10()`, `sin()`, `cos()`, `tan()`, `acos()`, `asin()`, `atan()`, `pow()`, `exp()`, ...
- ***alloc.h***: định nghĩa các hàm vào ra cấp thấp gồm các hàm `open()`, `_open()`, `read()`, `_read()`, `close()`, `_close()`, `creat()`, `_creat()`, `creatnew()`, `eof()`, `filelength()`, `lock()`, ...





Thứ tự ưu tiên của các toán tử

Mức	Toán tử							Thứ tự
Cao nhất	::							Cả hai
	()	[]	->	.			Trái tới phải	
	+ -	++ --	! ~	* &	new delete	sizeof ()	Phải tới trái	
	->*	.*					Trái tới phải	
	*	/	%				Trái tới phải	
	+	-					Trái tới phải	
	<<	>>					Trái tới phải	
	<	<=	>	>=			Trái tới phải	
	==	!=					Trái tới phải	
	&						Trái tới phải	
	^						Trái tới phải	
							Trái tới phải	
	&&						Trái tới phải	
							Trái tới phải	
	? :							Trái tới phải
		=	+= -=	*= /=	^= % =	&= =	<<= >>=	Phải tới trái
Thấp nhất	,						Trái tới phải	



Các kiểu dữ liệu cơ bản

❖ Kiểu **char**:

- Biểu diễn 1 ký tự thuộc ASCII (thực chất là số nguyên từ 0 đến 255)
- Ví dụ : Ký tự ASCII
 - 0 048
 - A 065
 - a 097

❖ Kiểu **int**:

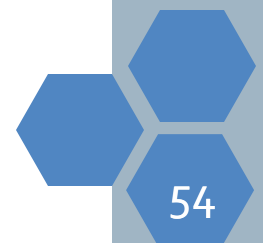
- 3 loại : int, long int (long) và unsigned int (unsigned).

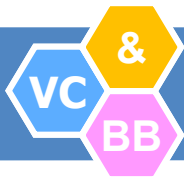
❖ Kiểu **float**:

- Biểu diễn các số thực độ chính xác đơn.

❖ Kiểu **double**:

- Biểu diễn các số thực độ chính xác kép.





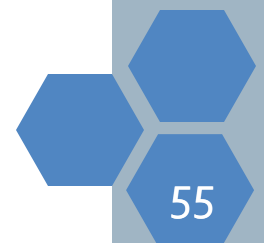
Các kiểu dữ liệu cơ bản

❖ Phạm vi, kích thước kiểu dữ liệu:

1. **char** 0..255 1byte
2. **int** -32768..32767 2bytes
3. **long** -2147483648..2147484647 4bytes
4. **unsigned** 0..65535 2bytes
5. **float** $3.4e - 38$.. $3.4e + 38$ 4bytes
6. **double** $1.7e - 308$.. $1.7e + 308$ 8bytes

❖ Kiểu void:

- Kiểu không giá trị, dùng để biểu diễn kết quả hàm cũng như nội dung của pointer.

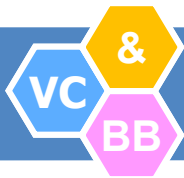


Cấu trúc (struct)

- ❖ Một cấu trúc dữ liệu là một tập hợp của những kiểu dữ liệu khác nhau được gộp lại với một cái tên duy nhất.
- ❖ Dạng thức

```
struct <tên_kiểu_cấu_trúc>
{
    // Khai báo các thành phần cấu trúc
    // ...
} <tên_cấu_trúc>;
```
- ❖ Khai báo biến, mảng cấu trúc:

```
tên_kiểu_cấu_trúc danh sách biến, mảng cấu trúc;
```

Cấu trúc (struct)

❖ Ví dụ:

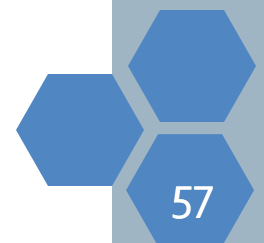
```
struct thi_sinh
{
    char hoten[25];
    long sobaodanh;
    float diemToan, diemLy, diemHoa, trungbinh;
};
thi_sinh bachkhoa[2000], kinhhte[2000], caodang[2000];
```

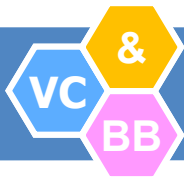
❖ Hoặc

```
struct thi_sinh
{
    char hoten[25];
    long sobaodanh;
    float diemToan, diemLy, diemHoa, trungbinh;
} bachkhoa[2000], kinhhte[2000], caodang[2000];
```

❖ Truy cập:

```
bachkhoa[5].hoten
kinhhte[12].diemToan
```





Trắc nghiệm

1

Khai báo biến nào sau đây **đúng**?

A

int length; float width;

B

int length, float width;

C

int length; width;

D

int length, int width;

2

Chọn kết quả in ra màn hình của đoạn chương trình sau:

A

x = 1, y = 2

B

x = 2, y = 1

C

x = 2, y = 2

D

Chương trình bị lỗi do x được khai báo lại

```
void main()
{
    int x = 1;
    int y = x = x + 1;
    printf("x = %d, y = %d", x, y);
}
```

3

Giả sử x là biến ký tự có giá trị 'b'.
Câu lệnh `printf("%c", ++x);` in ra:

A

a

B

b

C

c

D

d

4

Phân tích
đoạn mã sau:

```
#include <stdio.h>
void main()
{
    int i, j;
    printf("Enter an integer: ");
    scanf("%d", &j);
    i = i + 4;
}
```

A

Báo lỗi biên dịch vì i...

B

Báo lỗi biên dịch (error) vì j không được khởi tạo

C

Báo lỗi thực thi do i không có giá trị khởi tạo tại $i = i + 4$;

D

Chương trình biên dịch và thực thi bình thường

5

Giá trị cuối cùng của x bằng bao nhiêu khi x có giá trị khởi tạo là 1.

A

8

B

6

C

1

D

7

```
if (x >= 0)
    x += 5;
else if (x >= 5)
    x += 2;
```

6

Giả sử $x = 1, y = -1, z = 1$.

Cho biết kết quả in ra của đoạn chương trình sau:

A

$x > 0$ and $y > 0$

B

$x < 0$ and $z > 0$

C

$x < 0$ and $z < 0$

D

Không in gì cả

```
if (x > 0)
    if (y > 0)
        printf("x > 0 and y > 0");
else if (z > 0)
    printf("x < 0 and z > 0");
```

7

Cho biết giá trị cuối cùng của x.

A

1

B

2

C

3

D

4

```
int x = 3;  
if (x == 2);  
    x = 0;  
if (x == 3) x++;  
else x += 2;
```


8

y bằng bao nhiêu sau khi thực thi phát biểu switch sau:

A

1

B

2

C

3

D

4

```
int x = 3, y;  
switch (++x + 3) {  
    case 6: y = 0;  
    case 7: y = 1;  
    default: y += 1;  
}
```

9

Kết quả xuất ra của phát biểu switch dưới đây là gì?

A

abcd

B

bcd

C

bb

D

bbb

```
char ch = 'b';  
switch (ch) {  
    case 'a': printf("%c", ch);  
    case 'b': printf("%c", ch);  
    case 'c': printf("%c", ch);  
    case 'd': printf("%c", ch);  
}
```

10

Kết xuất cuối cùng của giá trị S ?

A

12

B

10

C

32

D

Lặp vô hạn

```
int S = 0, i = 1;
while (i = 1) {
    S = S + 2 * i;
    i++;
    if (i >= 5 || S > 30)
        break;
}
printf("%d", S);
```

11

Kết xuất cuối cùng của giá trị S?

A

45

B

55

C

50

D

Lặp vô hạn

```
int i = 1, S = 0;
while (1) {
    S += i++;
    if (i % 10 == 0)
        break;
}
printf("%d", S);
```

12

sum bằng bao nhiêu sau khi vòng lặp dưới đây kết thúc?

A

5

B

6

C

7

D

8

```
int sum = 0;
int item = 0;
do {
    item++;
    sum += item;
    if (sum > 4)
        break;
} while (item < 5);
```

13

Cho biết đoạn chương trình sau đây **xuất ra màn hình những gì?**

A

1 2

B

1 3

C

0 2

D

0 1

```
int i, a = 0;
for (i = 0; i < 3; i++) {
    if (i == 2)
        continue;
    a += i;
    if (i > 1) break;
    printf("%d ", a);
}
```

14

Kết quả cuối cùng của giá trị **S** sau vòng lặp?

A

60

B

14

C

5

D

32

```
int i, S = 0;
for (i = 5; i < 20; i += 2) {
    S += i;
    if ((i%5 == 0) && (i%3 == 0))
        break;
}
printf("%d", S);
```

15

Kết quả cuối cùng của giá trị **S** sau vòng lặp

A

32

B

8

C

5

D

Kết quả khác

```
int S = 0, i;  
for (i = 5; i < 20; i += 2) {  
    switch (i % 2) {  
        case 0: S += S; break;  
        case 1: S += 1; break;  
    }  
}  
printf("%d", S);
```


16

Kết quả của đoạn chương trình sau là gì?

A

5 10

B

6 12

C

4 5

D

Kết quả khác

```
int a[] = {1, 2, 3, 4, 5}, N = 5, i;  
for (i = 0; i < N/2; i++)  
    a[i] = a[N - i + 1];  
printf("%d %d", a[3], a[4]);
```

17

Kết quả của đoạn chương trình sau?

A

3 4 1 5 2

B

1 2 3 4 5

C

5 4 3 2 1

D

5 3 1 2 3

```
int a[] = {3, 4, 1, 5, 2}, N=5, x, y, t;  
for (x = 0; x < N; x++)  
    for (y = 0; y < N; y++)  
        if (a[x] > a[y]) {  
            t = a[x];  
            a[x] = a[y];  
            a[y] = t;  
        }  
for (x = 0; x < N; x++)  
    printf("%d ", a[x]);
```

18

Chức năng của đoạn chương trình sau là gì?

A

Kô thay đổi a

B

Sắp xếp a giảm

C

Đảo ngược a

D

Tìm max của a

```
int a[] = {3, 4, 1, 5, 2}, N = 5, i, k;  
for (i = 0; i < N/2; i++) {  
    k = a[i];  
    a[i] = a[N - i - 1];  
    a[N - i - 1] = k;  
}  
for (i = 0; i < N; i++)  
    printf("%d ", a[i]);
```

1

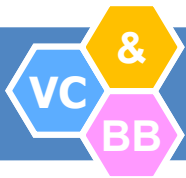
Viết hàm tính và trả về số ngày của một tháng thuộc một năm cho trước (không cần kiểm tra tính hợp lệ của tháng và năm cho trước này). Nguyên mẫu hàm như sau:

```
int TinhSoNgayTrongThang(int th, int nm);
```

3

Viết hàm sắp xếp giảm dần mảng số nguyên a, số lượng phần tử n cho trước và trả về số lần đã thực hiện việc đổi chỗ (hoán vị).
Nguyên mẫu hàm như sau:

```
int SapXepGiamDan(int a[], int n);
```

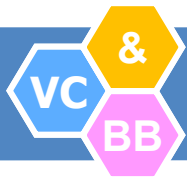


Tự luận – Bài 1

```
{
```

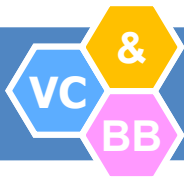
```
{
```

... ➔ trang kế tiếp



Tự luận – Bài 1 (tiếp theo)

}



Tự luận – Bài 3

```
int SapXepGiamDan(int a[], int n)
```

```
{
```

```
}
```