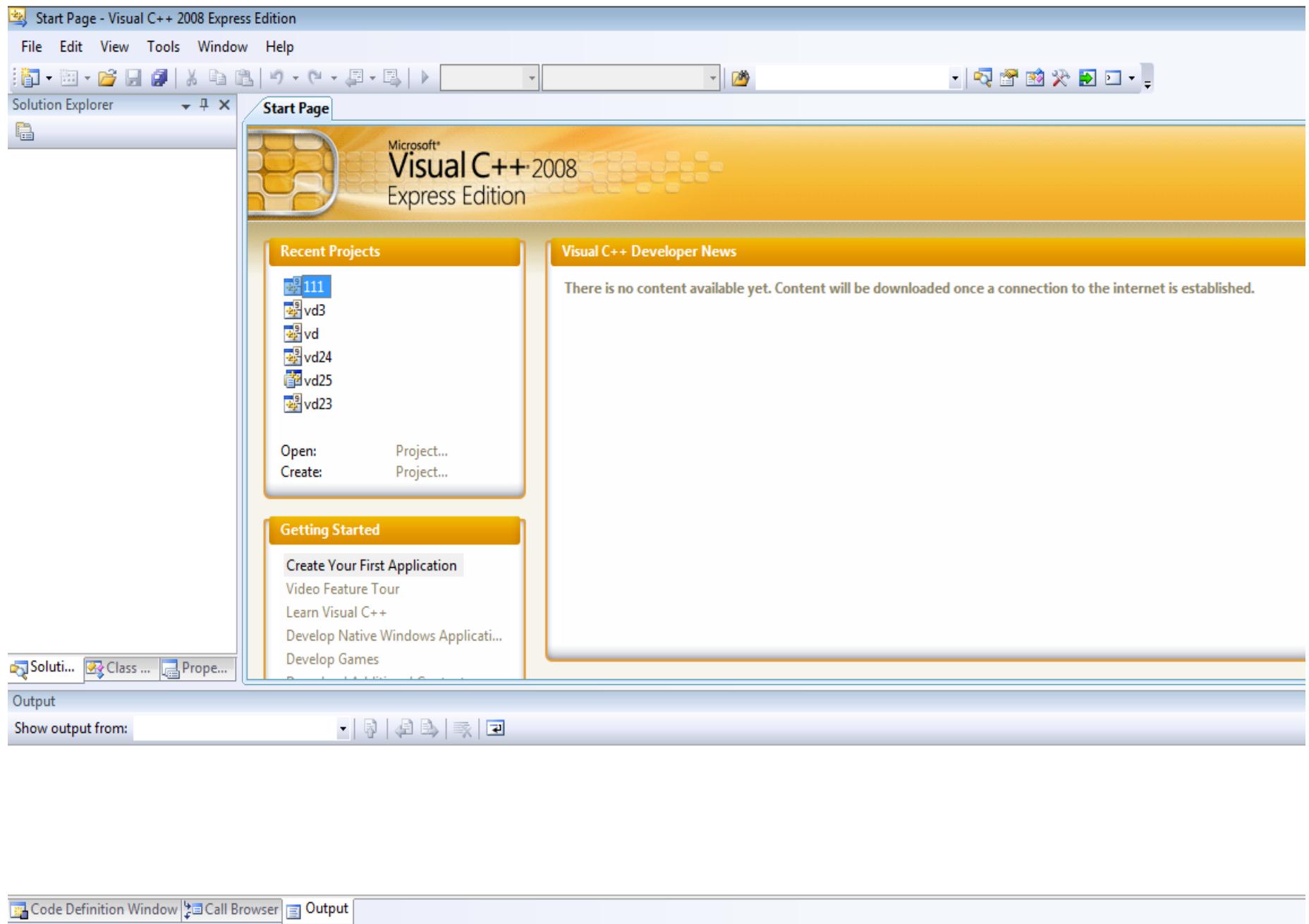


Hướng dẫn tạo file trong Visual C++

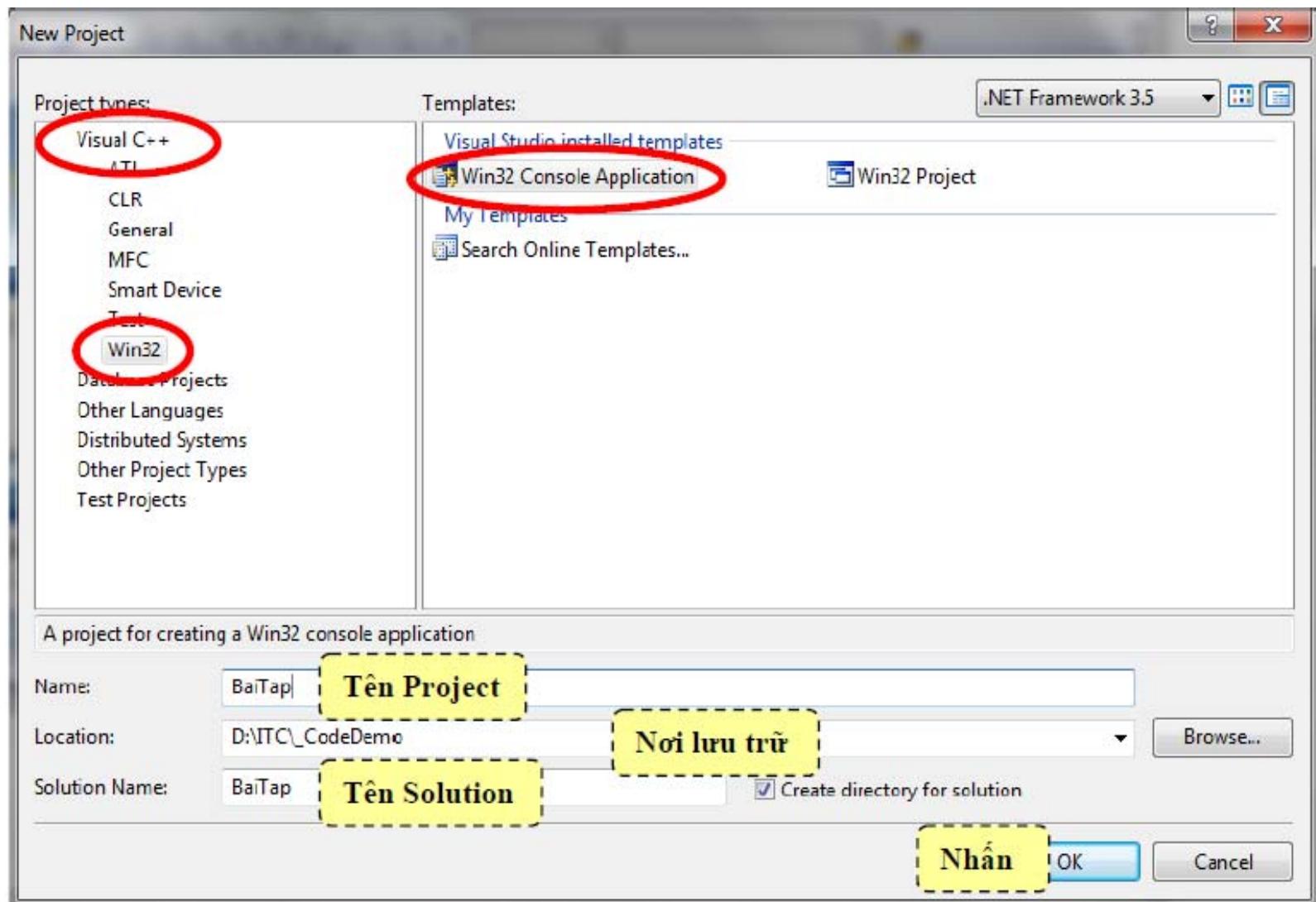
GV: Võ Thị Ngọc Thủy

Tạo mới Project trong VC++ 2008

- Bước 1: Khởi động VC++2008
- Xuất hiện giao diện đầu tiên: Start page



- Tạo Project mới: File \ New \ Project...
- Xuất hiện cửa sổ New project



- Tab Projects chọn Win32 Console Application
- Project name: Nhập tên của dự án (Ví dụ đặt tên là BaiTap1)
- Location: Vị trí lưu dự án

VC sẽ tạo một thư mục với tên là tên Project tại vị trí đã chỉ định. Bên trong thư mục này sẽ chứa toàn bộ các file liên quan đến Project. Nếu muốn sao chép Project thì chỉ việc chép thư mục này.

Win32 Application Wizard - BaiTap



Welcome to the Win32 Application Wizard

Overview

Application Settings

These are the current project settings:

- Console application

Click **Finish** from any window to accept the current settings.

After you create the project, see the project's `readme.txt` file for information about the project features and files that are generated.

< P

Nhấn

Next >

Finish

Cancel

Win32 Application Wizard - BaiTap

Application Settings



Overview

Application Settings

Application type:

- Windows application
- Console application
- DLL
- Static library

Additional options:

- Empty project
- Export symbols
- Precompiled header

Add common header files for:

- ATL
- MFC

< Previous

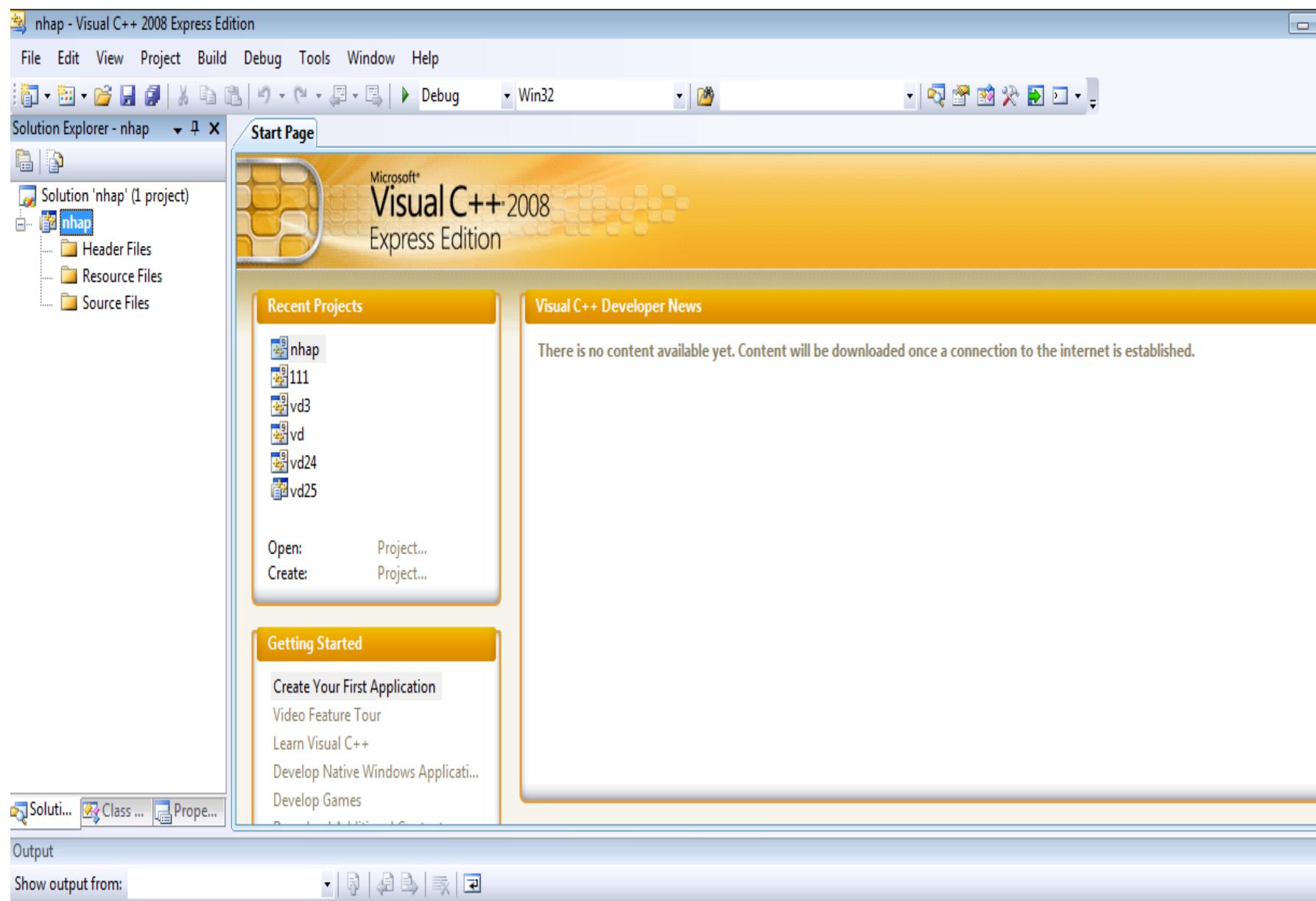
Next >

Nhấn

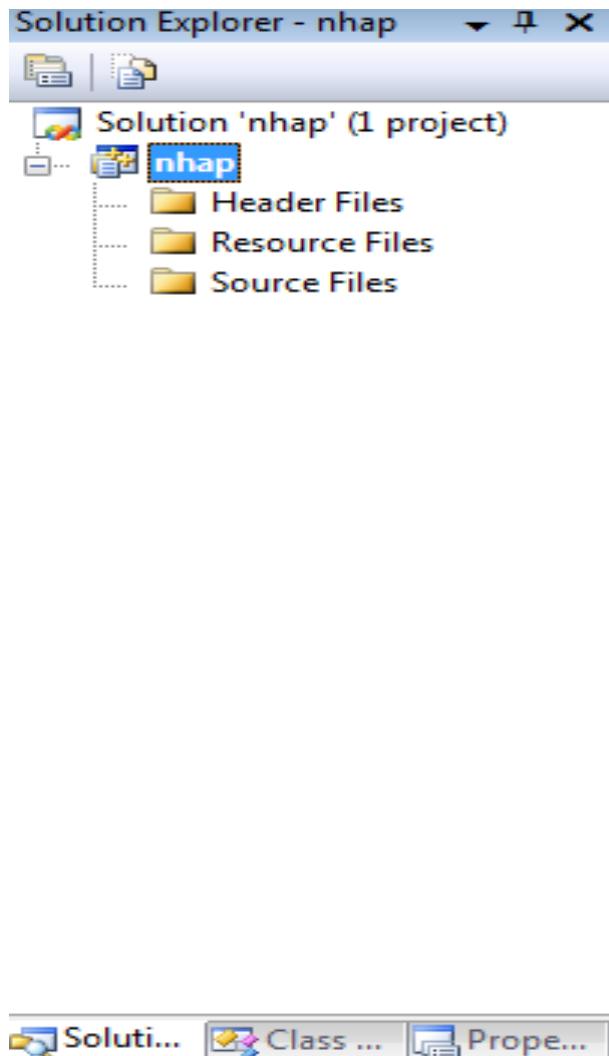
Finish

Cancel

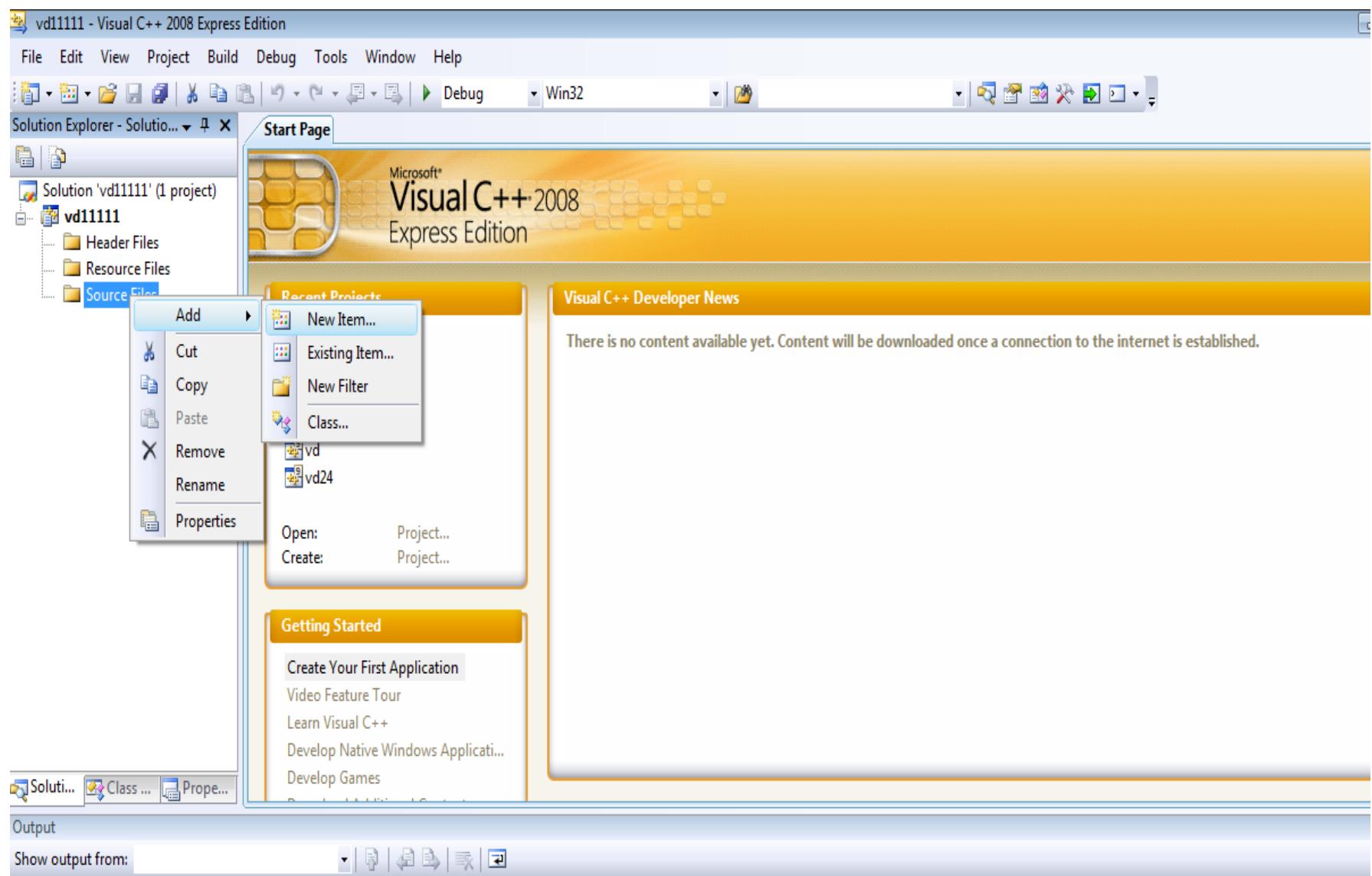
Giao diện sau khi tạo Project

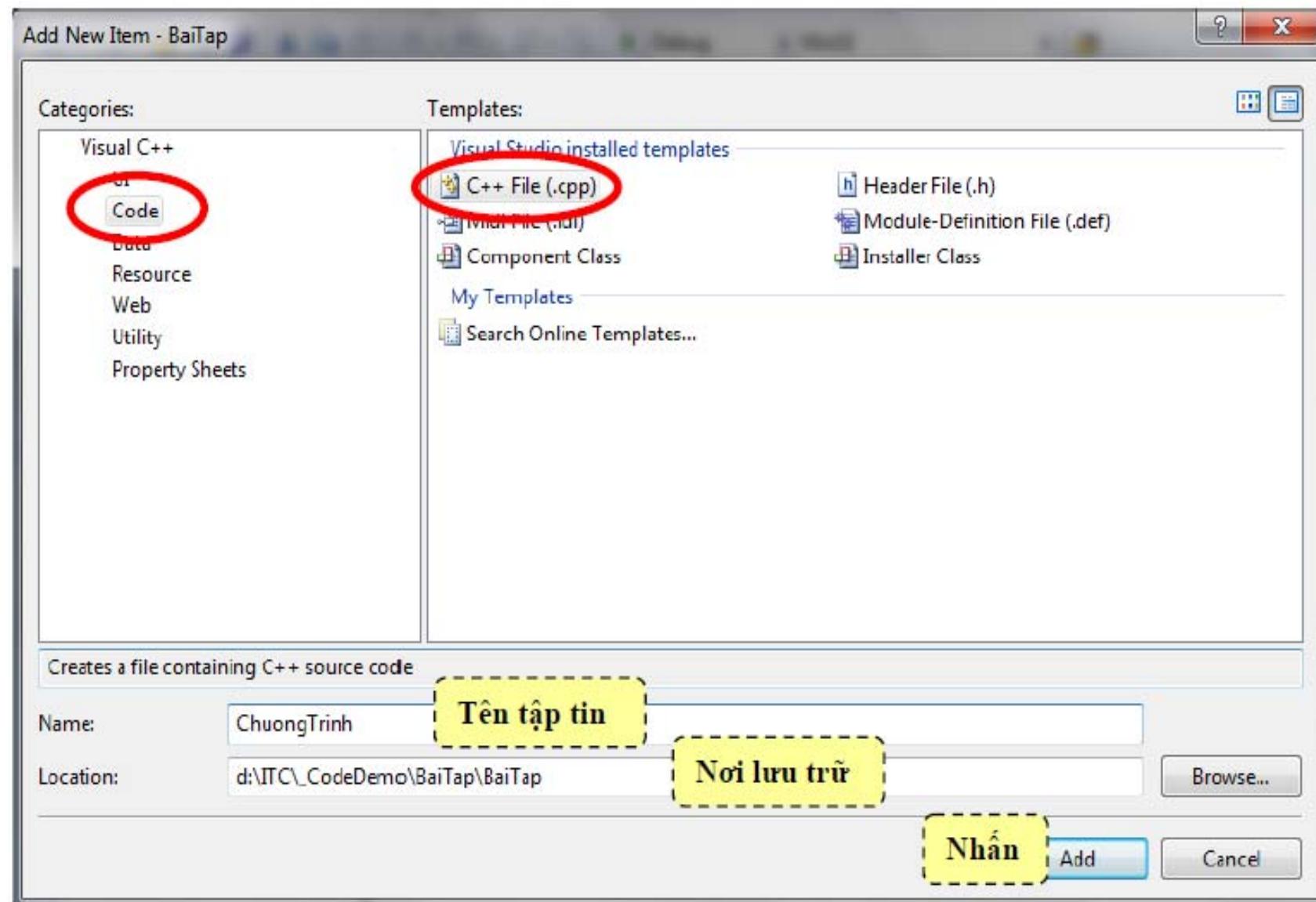


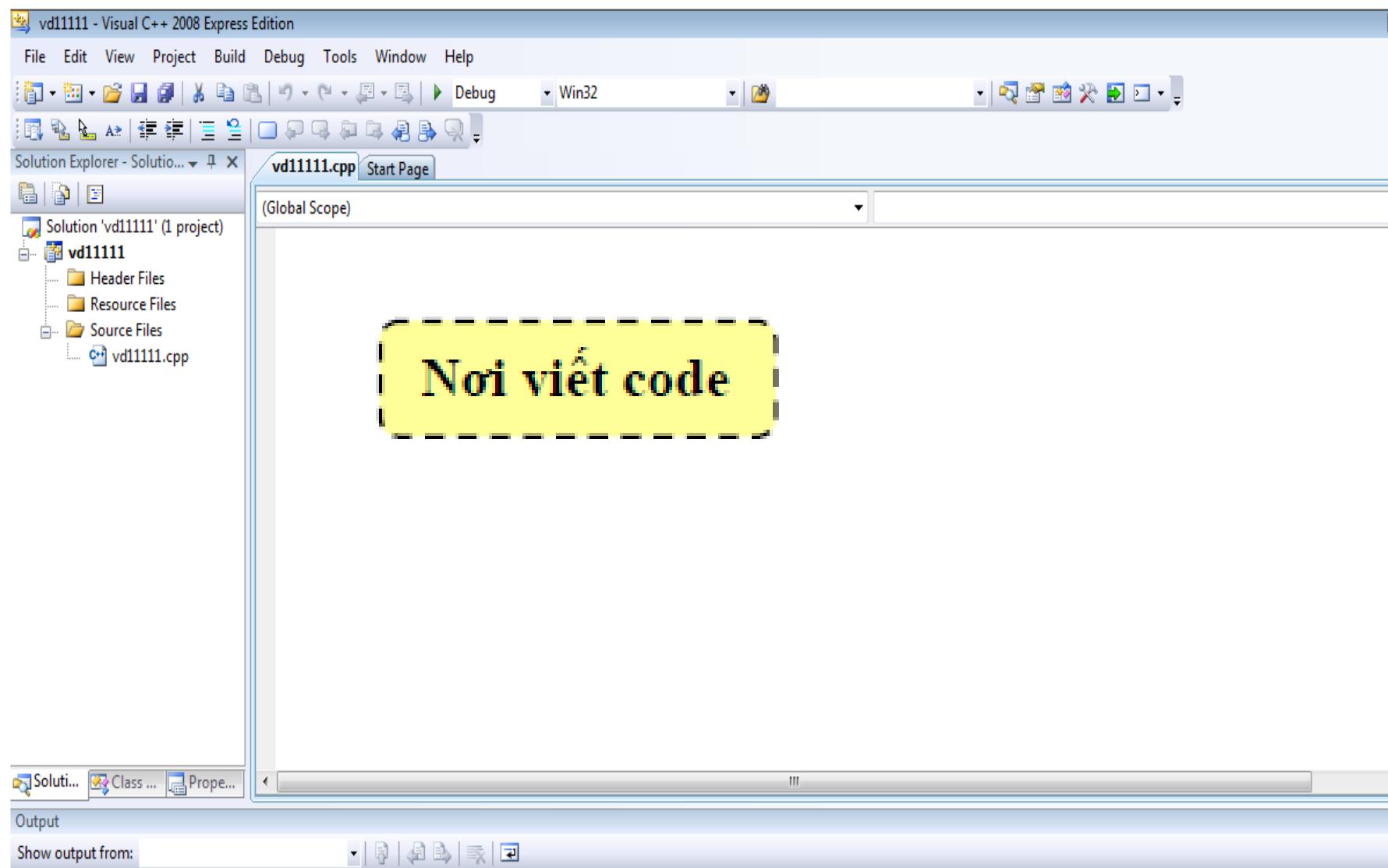
Mô tả cửa sổ Workspace



- Source Files (*.cpp): Chứa các file cài đặt các hàm
- Header Files (*.h): Chứa các file khai báo (hàm, biến toàn cục, kiểu cấu trúc, ...)
- Resource Files: Chứa các tài nguyên (icon, bitmap, dialog, ...). **Đối với lập trình ứng dụng Console thì không dùng đến.**

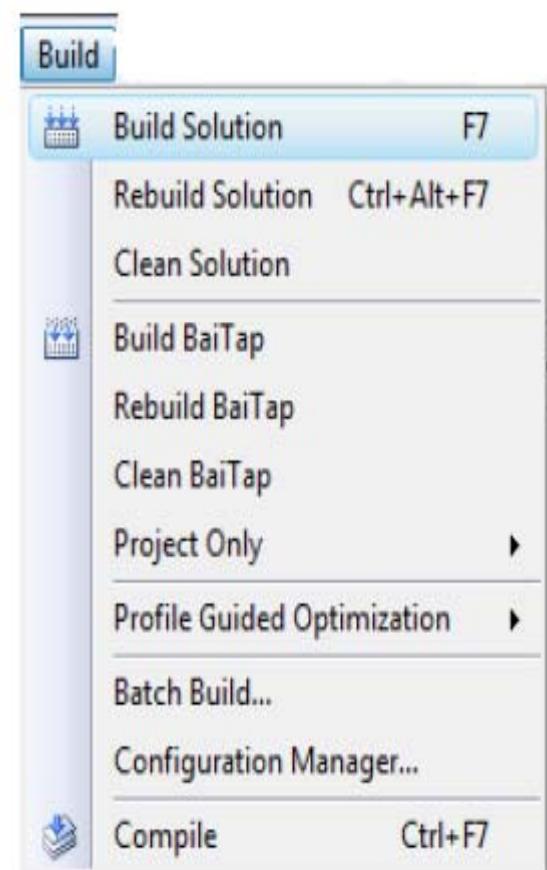






Dịch, sửa lỗi và chạy chương trình

- Biên dịch và kiểm tra lỗi:
Nhấn F7
- Nếu có thông báo lỗi (error) thì
nhấn F4 để tìm, quan sát và
sửa lỗi (cửa sổ lỗi xuất hiện
bên dưới màn hình soạn thảo
code)
- Chạy chương trình: Ctrl + F5



Output

Show output from: Build

```
1>----- Build started: Project: BaiTap, Configuration: Debug Win32 -----  
1>Compiling...  
1>ChuongTrinh.cpp  
1>d:\itc\_codedemo\baitap\baitap\chuongtrinh.cpp(8) : error C2143: syntax error : missing ';' before ')'  
1>Build log was saved at "file:///d:/ITC/_CodeDemo/baitap/BaiTap/Debug/BuildLog.htm"  
1>BaiTap - 1 error(s), 0 warning(s)  
===== Build: 0 succeeded, 1 failed, 0 up-to-date, 0 skipped =====
```

Nếu chương trình không bị lỗi CÚ PHÁP: cửa sổ Output báo thành công

Output

Show output from: Build

```
1>Copyright (C) Microsoft Corporation. All rights reserved.  
1>Linking...  
1>Embedding manifest...  
1>Microsoft (R) Windows (R) Resource Compiler Version 6.0.5724.0  
1>Copyright (C) Microsoft Corporation. All rights reserved.  
1>Build log was saved at "file:///d:/ITC/_CodeDemo/BaiTap/BaiTap/Debug/BuildLog.htm"  
1>BaiTap - 0 error(s), 0 warning(s)  
===== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped =====
```

Mở Project có sẵn

- **Cách 1: Mở từ menu VC**
 - Khởi động VC
 - Vào menu File/ Open
- Chọn thư mục chứa Project, chọn tên Project, sau đó chọn Open

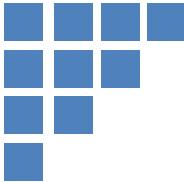
Cách 2: Mở thư mục chứa Project

Trường ĐH Khoa Học Tự Nhiên
Khoa Vật Lý – Bộ Môn Vật Lý Ứng Dụng



NGÔN NGỮ LẬP TRÌNH C++

CÁC KHÁI NIỆM CƠ BẢN



Các khái niệm cơ bản

- **Lập trình máy tính**

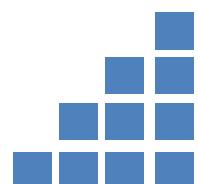
- Gọi tắt là **lập trình** (programming).
- Nghệ thuật **cài đặt** một hoặc nhiều **thuật toán** trừu tượng có liên quan với nhau bằng một **ngôn ngữ lập trình** để tạo ra một **chương trình máy tính**.

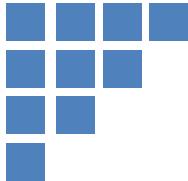
- **Ngôn ngữ lập trình**

Là hệ thống các ký hiệu tuân theo các qui ước về ngữ pháp và ngữ nghĩa, dùng để xây dựng thành các chương trình

- **Thuật toán**

- Là **tập hợp** (dãy) **hữu hạn** các **chỉ thị** (hành động) được **định nghĩa rõ ràng** nhằm **giải quyết** một bài toán **cụ thể** nào đó.

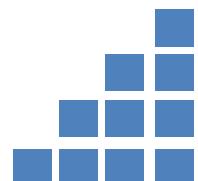


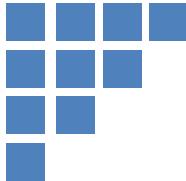


Các tính chất của thuật toán

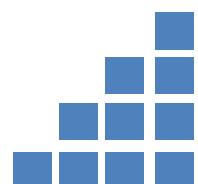
- **Bao gồm 5 tính chất sau:**

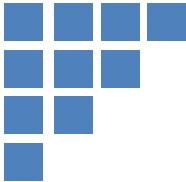
- **Tính chính xác:** quá trình tính toán hay các thao tác máy tính thực hiện là chính xác.
- **Tính rõ ràng:** các câu lệnh minh bạch được sắp xếp theo thứ tự nhất định.
- **Tính khách quan:** được viết bởi nhiều người trên máy tính nhưng kết quả phải như nhau.
- **Tính phổ dụng:** có thể áp dụng cho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





Các bước xây dựng chương trình



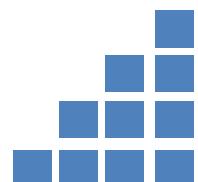


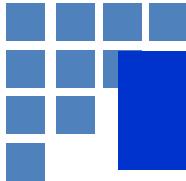
Sử dụng ngôn ngữ tự nhiên

Đầu vào: a, b thuộc \mathbb{R}

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

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





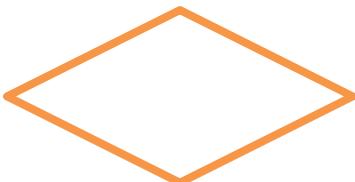
Sử dụng lưu đồ - sơ đồ khối



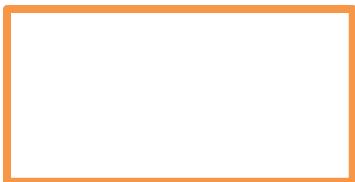
Khối giới hạn
Chỉ thị bắt đầu và kết thúc.



Khối vào ra
Nhập/Xuất dữ liệu.



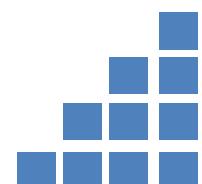
Khối lựa chọn
Tùy điều kiện sẽ rẽ nhánh.



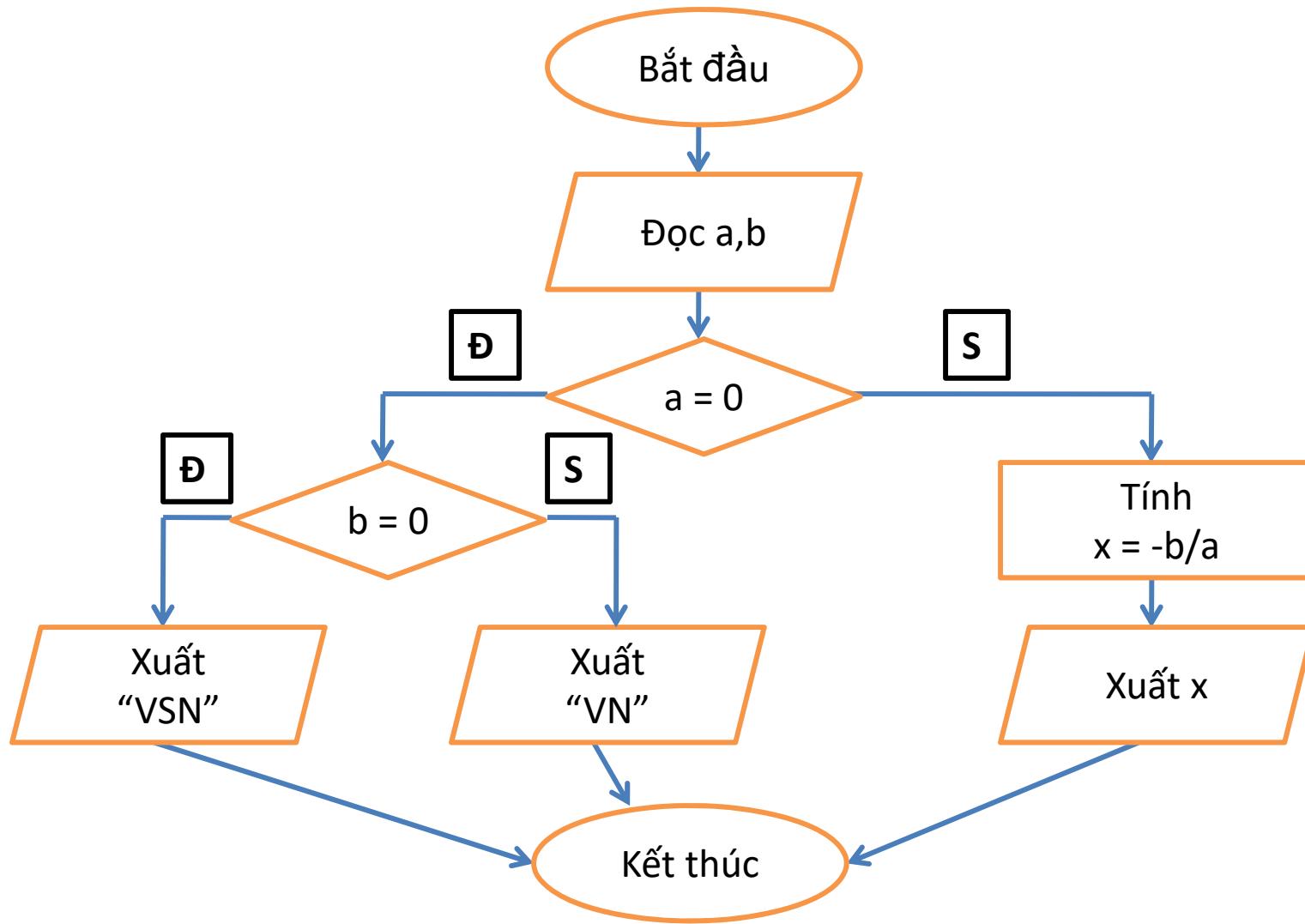
Khối thao tác
Ghi thao tác cần thực hiện.

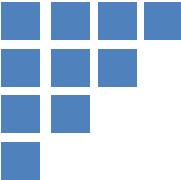


Đường đi
Chỉ hướng thao tác tiếp theo.



Sử dụng lưu đồ - sơ đồ khối





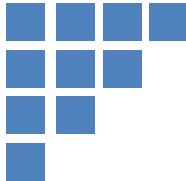
Sử dụng mã giả

- Vay mượn ngôn ngữ nào đó (ví dụ Pascal) để biểu diễn thuật toán.

Đầu vào: a, b thuộc \mathbb{R}

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

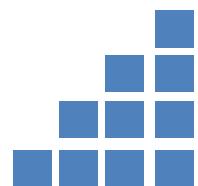
```
If a = 0 Then  
Begin  
    If b = 0 Then  
        Xuất "Phương trình vô số nghiệm"  
    Else  
        Xuất "Phương trình vô nghiệm"  
End  
Else  
    Xuất "Phương trình có nghiệm  $x = -b/a$ "
```

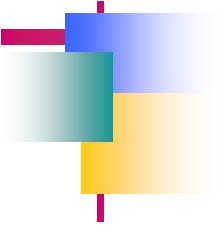


Cài đặt thuật toán bằng C/C++

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

void main()
{
    int a, b;
    cin>> "Nhập a, b: ";
    cout<<a<<b;
    if (a == 0)
        if (b == 0)
            cout<<"Phương trình VSN";
        else
            cout<<"Phương trình VN";
    else
        cout<<"x = %.2f", -float(b) / a;
}
```





KỸ THUẬT LẬP TRÌNH C++

CÁC KIỀU DỮ LIỆU CƠ SỞ

GV: VÕ THỊ NGỌC THUÝ

Nội dung

1 Các kiểu dữ liệu cơ sở

2 Biến, Hằng, Phép toán, Biểu thức
& Câu lệnh

3 Các lệnh nhập xuất

4 Một số ví dụ minh họa

1. Các kiểu dữ liệu cơ sở

- C++ có 4 kiểu cơ sở như sau:
 - Kiểu số nguyên: giá trị của nó là các số nguyên như 2912, -1706, ...
 - Kiểu số thực: giá trị của nó là các số thực như 3.1415, 29.12, -17.06, ...
 - Kiểu luận lý: giá trị đúng hoặc sai.
 - Kiểu ký tự: 256 ký tự trong bảng mã ASCII.

Kiểu số nguyên

- **Các kiểu số nguyên (có dấu)**

- n bit có dấu: $-2^{n-1} \dots +2^{n-1} - 1$

Kiểu (Type)	Độ lớn (Byte)	Miền giá trị (Range)	Cách sử dụng
char	1	-128 ... +127	Ký tự đơn "a", "1", ...
int	2	-32.768 ... +32.767	Số nguyên có dấu
short	2	-32.768 ... +32.767	Số nguyên ngắn
long	4	-2.147.483.648 ... +2.147.483.647	Số nguyên dài

Kiểu số nguyên

- **Các kiểu số nguyên (không dấu)**

- n bit không dấu: $0 \dots 2^n - 1$

Kiểu (Type)	Độ lớn (Byte)	Miền giá trị (Range)
unsigned char	1	0 ... 255
unsigned int	2	0 ... 65.535
unsigned short	2	0 ... 65.535
unsigned long	4	0 ... 4.294.967.295

Kiểu số thực

- **Các kiểu số thực (floating-point)**

- Ví dụ

- $17.06 = 1.706 \cdot 10 = 1.706 \cdot 10^1$

Kiểu (Type)	Độ lớn (Byte)	Miền giá trị (Range)
float (*)	4	$3.4 \cdot 10^{-38} \dots 3.4 \cdot 10^{38}$
double (**)	8	$1.7 \cdot 10^{-308} \dots 1.7 \cdot 10^{308}$

- (*) Dấu phẩy động đơn- độ chính xác đơn (Single-precision) chính xác đến 7 số lẻ.
- (**) Dấu phẩy động kép - độ chính xác kép (Double-precision) chính xác đến 19 số lẻ.

Kiểu luận lý

- **Đặc điểm**

- C ngầm định một cách không tường minh:
 - **false** (sai): giá trị 0.
 - **true** (đúng): giá trị khác 0, thường là 1.
- C++: **bool**

- **Ví dụ**

- 0 (false), 1 (true), 2 (true), 2.5 (true)
- $1 > 2$ (0, false), $1 < 2$ (1, true)

Kiểu ký tự

- **Đặc điểm**

- Tên kiểu: **char**
- Miền giá trị: 256 ký tự trong bảng mã ASCII.
- Chính là kiểu số nguyên do:
 - Lưu tất cả dữ liệu ở dạng số.
 - Không lưu trực tiếp ký tự mà chỉ lưu mã ASCII của ký tự đó.

- **Ví dụ**

- Lưu số 65 tương đương với ký tự 'A'...
- Lưu số 97 tương đương với ký tự 'a'.

Biến

Chương trình sử dụng biến để
lưu trữ dữ liệu

Biến

Ví dụ về khai báo biến không khởi tạo

```
void main()
{
    int i, j; // khai báo 2 biến i, j có kiểu nguyên
    float x; // khai báo biến thực x
    char c, d[100]; // biến kí tự c, xâu d
                    // chứa tối đa 100 kí tự
    unsigned int u; // biến nguyên không dấu u
    ...
}
```

Cú pháp khai báo biến không khởi tạo:
<tên kiểu> <tên biến>;

<tên kiểu> <tên biến 1>, <tên biến 2>;

Cú pháp khai báo biến có khởi tạo:

<tên kiểu> < tên_bien = gt>;
<tên kiểu> < tên_bien_1 = gt_1 >, < tên_bien_2 = gt_2 >;

Ví dụ:

Ví dụ về khai báo biến có khởi tạo

```
const int n = 10 ;
void main()
{
    int i = 2, j , k = n + 5;      // khai báo i và khởi tạo
                                    // bằng 2, k bằng 15
    float eps = 1.0e-6 ;          // khai báo biến thực
                                    // epsilon khởi tạo bằng 10-6
    char c = 'Z';                // khai báo biến kí tự c
                                    // và khởi tạo bằng 'A'
    char d[100] = "Tin hoc";    // khai báo xâu kí tự d
                                    // chứa dòng chữ "Tin hoc"
```



Hằng số

Hằng thường

Là các giá trị cố định, được đặt tên gọi trong chương trình C/C++

Cú pháp
`<kiểu> <tênhằng> = <giá trị>;`

Ví dụ

```
int a = 1506;           // 150610
int b = 01506;          // 15068
int c = 0x1506;         // 150616 (0x hay 0X)
float d = 15.06e-3;    // 15.06*10-3 (e hay E)
```

Hằng số

Hằng ký hiệu

Là hằng được gán tên trong chương trình,
và không thay đổi giá trị

Cú pháp

#define <tên hằng> <giá trị>

hoặc

const <tên kiểu> <tên hằng> = <giá trị>

Ví dụ

```
#define MAX 100
#define PI 3.14
const int MAX = 100;
const float PI = 3.14;
```

// Không có ;

// Không có ;

Phép toán, biểu thức, câu lệnh



Phép toán

- C++ có nhiều phép toán chia thành các loại 1 ngôi, 2 ngôi và thậm chí 3 ngôi
- Các thành phần tên gọi tham gia trong phép toán gọi là hạng thức hoặc toán hạng, các kí hiệu phép toán gọi là toán tử
- Ví dụ: $a+b$: a , b là toán hạng, $+$ là toán tử
- Số ngôi của phép toán chính là số toán hạng

Các toán tử toán học

• Toán tử 1 ngôi

- Chỉ có một toán hạng trong biểu thức.
- Có hai toán hạng 1 ngôi:
++ (tăng 1 đơn vị), -- (giảm 1 đơn vị)
 - + Đặt trước toán hạng → toán hạng sẽ tăng/giảm trước khi sử dụng
 - Ví dụ ++x hay --x: thực hiện tăng/giảm trước.
 - + Đặt sau toán hạng → toán hạng được sử dụng rồi mới tăng/giảm
 - Ví dụ x++ hay x--: thực hiện tăng/giảm sau.

• Ví dụ

- $x = 10; y = x++;$ // $y = 10$ và $x = 11$ (gán x vào y, sau đó tăng y)
- $x = 10; y = ++x;$ // $x = 11$ và $y = 11$

Các toán tử toán học (tt)

- **Toán tử 2 ngôi**

- Có hai toán hạng trong biểu thức.
- Có 5 toán tử toán học 2 ngôi: +, -, *, /, %
(chia lấy phần dư)
 - $x = x + y \Leftrightarrow x += y;$

- **Ví dụ**

- $a = 1 + 2; b = 1 - 2; c = 1 * 2; d = 1 / 2;$
- $e = 1 * 1.0 / 2; f = \text{float}(1) / 2; g = \text{float}(1 / 2);$
- $h = 1 \% 2;$
- $x = x * (2 + 3 * 5); \Leftrightarrow x *= 2 + 3 * 5;$

Toán tử gán

- **Khái niệm**

- Thường được sử dụng trong lập trình.
- Gán giá trị cho biến.

- **Cú pháp: Gán thông thường**

- `<biến> = <giá trị>;`
- `<biến> = <biến>;`
- `<biến> = <biểu thức>;`

- * Có thể thực hiện liên tiếp phép gán.

- **Ví dụ**

```
void main()
{
    int a, b, c, d, e, thuong;
    a = 10;
    b = a;
    thuong = a / b;
    a = b = c = d = e = 156;
    e = 156;
    d = e;
    c = d;
    b = c;
    a = b;
}
```

Gán có điều kiện: toán tử 3 ngôi

<biến> = <điều kiện>?<biểu thức 1>:<biểu thức 2>

- <điều kiện> là một biểu thức logic.
- <biểu thức 1> và <biểu thức 2> là các biểu thức cùng kiểu với kiểu của <biến>
- Nếu <điều kiện> đúng thì <biến> nhận giá trị của <biểu thức 1> ngược lại nhận giá trị của <biểu thức 2>

Ví dụ gán có điều kiện:

1. $x = (3 + 4 < 7) ? 10: 20 // x = 20$ vì $3+4<7$ là sai
2. $x = (3 + 4) ? 10: 20 // x = 10$ vì $3+4$ khác 0, tức điều kiện đúng
3. $x = (a > b) ? a: b // x =$ số lớn nhất trong 2 số a, b.

Các toán tử trên bit

- **Các toán tử trên bit**

- Tác động lên các bit của toán hạng (nguyên).
- **&** (and), **|** (or), **^** (xor), **~** (not hay lấy số bù 1)
- **>>** (shift right), **<<** (shift left)
- Toán tử gộp: **&=**, **|=**, **^=**, **~=**, **>>=**, **<<=**

&	0	1
0	0	0
1	0	1

^	0	1
0	0	1
1	1	0

	0	1
0	0	1
1	1	1

~	0	1
1	1	0

Các toán tử trên bit

- **Ví dụ**

```
void main()
{
    int a = 5;    // 0000 0000 0000 0101
    int b = 6;    // 0000 0000 0000 0110

    int z1, z2, z3, z4, z5, z6;
    z1 = a & b; // 0000 0000 0000 0100
    z2 = a | b; // 0000 0000 0000 0111
    z3 = a ^ b; // 0000 0000 0000 0011
    z4 = ~a;     // 1111 1111 1111 1010
    z5 = a >> 2;// 0000 0000 0000 0001
    z6 = a << 2;// 0000 0000 0001 0100
}
```

Các toán tử quan hệ

- **Các toán tử quan hệ**

- So sánh 2 biểu thức với nhau
- Cho ra kết quả 0 (hay false nếu sai) hoặc 1 (hay true nếu đúng)
 - `==`, `>`, `<`, `>=`, `<`, `<=`, `!=`

- **Ví dụ**

- `s1 = (1 == 2);` `s2 = (1 != 2);`
- `s3 = (1 > 2);` `s4 = (1 >= 2);`
- `s5 = (1 < 2);` `s6 = (1 <= 2);`

Các toán tử luận lý

• Các toán tử luận lý

- Tổ hợp nhiều biểu thức quan hệ với nhau thành biểu thức đơn, và có thể xác định tính đúng/sai của biểu thức này.
- Các toán tử luận lý: **&&** (and), **||** (or), **!** (not)

&&	0	1
0	0	0
1	0	1

	0	1
0	0	1
1	1	1

- Ví dụ
 - $s1 = (1 > 2) \&\& (3 > 4);$
 - $s2 = (1 > 2) || (3 > 4);$
 - $s3 = !(1 > 2);$

Toán tử phẩy

- **Toán tử phẩy**

- Các biểu thức đặt cách nhau bằng dấu ,
- Các biểu thức con **lần lượt được tính từ trái sang phải.**
- Biểu thức mới nhận được là **giá trị của biểu thức bên phải cùng.**

- **Ví dụ**

- $x = (a++, b = b + 2);$
- $a++; b = b + 2; x = b;$

Thứ tự ưu tiên của các toán tử

C++ qui định trật tự tính toán theo các mức độ ưu tiên từ cao đến thấp như sau:

1. Các biểu thức trong cặp dấu ngoặc ()
2. Nếu có nhiều cặp ngoặc lồng nhau thì cặp trong cùng (sâu nhất) được ưu tiên cao hơn.
3. Các phép toán 1 ngôi (tự tăng-giảm, lấy địa chỉ, lấy nội dung con trỏ, phủ định ...)
4. Các phép toán số học.
5. Các phép toán quan hệ, logic.
6. Các phép gán.

Thư viện các hàm toán học

#include <math.h>

- `abs(x), labs(x), fabs(x)`: trả lại giá trị tuyệt đối của một số nguyên, số nguyên dài và số thực.
- `pow(x, y)`: hàm mũ, trả lại giá trị x lũy thừa y (x^y).
- `exp(x)`: hàm mũ, trả lại giá trị e mũ x (e^x).
- `log(x), log10(x)`: trả lại lôgarit cơ số e và lôgarit thập phân của x ($\ln x, \log x$) .
- `sqrt(x)`: trả lại căn bậc 2 của x .
- `atof(s_number)`: trả lại số thực ứng với số viết dưới dạng xâu kí tự `s_number`.
- Hàm lượng giác: `sin(x), cos(x), tan(x)`

Biểu thức

- **Khái niệm**

- Biểu thức là dãy kí hiệu kết hợp giữa các toán hạng (**operand**), toán tử (**operator**) và cặp dấu () theo một qui tắc nhất định
- Toán tử tác động lên các giá trị của toán hạng và cho giá trị có kiểu nhất định.
- Toán tử: +, -, *, /, %....
- Toán hạng: **hằng, biến, hàm...**

- **Ví dụ**

- $2 + 3$, $a / 5$, $(a + b) * 5$, ...

Viết biểu thức cho các mệnh đề

- **x lớn hơn hay bằng 3**

$$x \geq 3$$

- **a và b cùng dấu**

$$((a>0) \&\& (b>0)) \text{ || } ((a<0) \&\& (b<0))$$

$$(a>0 \&\& b>0) \text{ || } (a<0 \&\& b<0)$$

- **p bằng q bằng r**

$$(p == q) \&\& (q == r) \text{ hoặc } (p == q \&\& q == r)$$

- **$-5 < x < 5$**

$$(x > -5) \&\& (x < 5) \text{ hoặc } (x > -5 \&\& x < 5)$$

Câu lệnh

- **Khái niệm**

- Là một chỉ thị trực tiếp, hoàn chỉnh nhằm ra lệnh cho máy tính thực hiện một số tác vụ nhất định nào đó.
- Một câu lệnh trong C++ được thiết lập từ các từ khoá và các biểu thức ... và luôn luôn được kết thúc bằng dấu chấm phẩy

- **Ví dụ:**

- `cin >> x >> y ;`
- `x = 3 + x ; y = (x = sqrt(x)) + 1 ;`
- `cout << x ;`
- `cout << y ;`

Câu lệnh

- **Phân loại**

- Câu lệnh đơn: chỉ gồm một câu lệnh.
- Câu lệnh phức (khối lệnh): gồm nhiều câu lệnh đơn được bao bởi { và }

- **Ví dụ**

```
a = 2912; // Câu lệnh đơn
```

```
{ // Câu lệnh phức/khối lệnh
    a = 2912;
    b = 1706;
}
```

Câu lệnh nhập

- **Thư viện**

```
#include <iostream>
using namespace std;
```

- **Cú pháp**

```
cin >> biến_1 ;
cin >> biến_2 ;
cin >> biến_3 ;
```

hoặc:

```
cin >> biến_1 >> biến_2 >> biến_3 ;
```

- Muốn có khoảng cách giữa các biến thì ta khai báo

include <iomanip>

Using std::setw;

cin>> biến 1 >>setw() >>biến 2;

Chú ý: toán tử nhập >> chủ yếu làm việc với dữ liệu kiểu số. Để nhập kí tự hoặc xâu kí tự, C++ cung cấp các phương thức (hàm) sau:

- **cin.get(c):** cho phép nhập một kí tự vào biến kí tự c,
- **cin.getline(s,n):** cho phép nhập tối đa n-1 kí tự vào xâu s.

Câu lệnh xuất

- Trong C++ để xuất giá trị của các **biểu thức** ra màn hình ta dùng câu lệnh sau:

```
cout << bt_1 ; cout << bt_2 ; cout << bt_3 ;
```

- hoặc:

```
cout << bt_1 << bt_2 << bt_3 ;
```

Bài tập thực hành

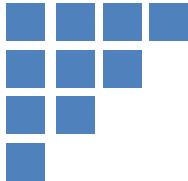
1. Nhập 2 số a và b. Tính tổng, hiệu, tích và thương của hai số đó.
2. Nhập tên sản phẩm, số lượng và đơn giá. Tính tiền và thuế giá trị gia tăng phải trả, biết:
 - a. tiền = số lượng * đơn giá
 - b. thuế giá trị gia tăng = 10% tiền
3. Viết chương trình cho phép nhập vào chiều rộng và chiều cao của một hình chữ nhật. In ra chu vi và diện tích của hình chữ nhật đó.
4. Viết chương trình cho phép nhập vào 3 cạnh của một tam giác. In ra diện tích và chu vi của tam giác đó.
5. Viết chương trình cho phép nhập vào giờ, phút, giây. In ra tổng số giây tương ứng.
6. Viết chương trình cho phép nhập vào cơ số a và số mũ x. Tính và in ra a^x và $\log_a x$.
7. Viết chương trình cho phép nhập vào một số dương A và một số nguyên n. Tính và in ra căn bậc n của số A.

Trường ĐH Khoa Học Tự Nhiên
Khoa Vật Lý – Bộ Môn Vật Lý Ứng Dụng

KỸ THUẬT LẬP TRÌNH C++

Chương 2: CẤU TRÚC ĐIỀU KHIỂN

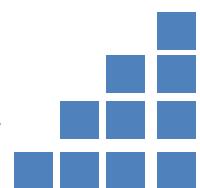
Phần 1: Cấu trúc điều kiện và rẽ nhánh

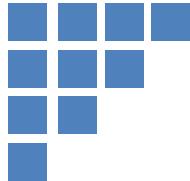


MỤC TIÊU

Sau khi hoàn tất bài này học viên sẽ hiểu và vận dụng các kiến thức kĩ năng cơ bản sau:

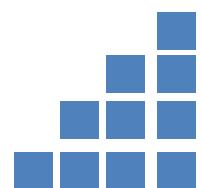
- Cú pháp, ý nghĩa, cách sử dụng lệnh if, lệnh switch.
- Một số bài toán sử dụng lệnh if, switch thông qua các ví dụ.
- So sánh, đánh giá một số bài toán sử dụng lệnh if hoặc switch.
- Cách sử dụng các cấu trúc lồng nhau.





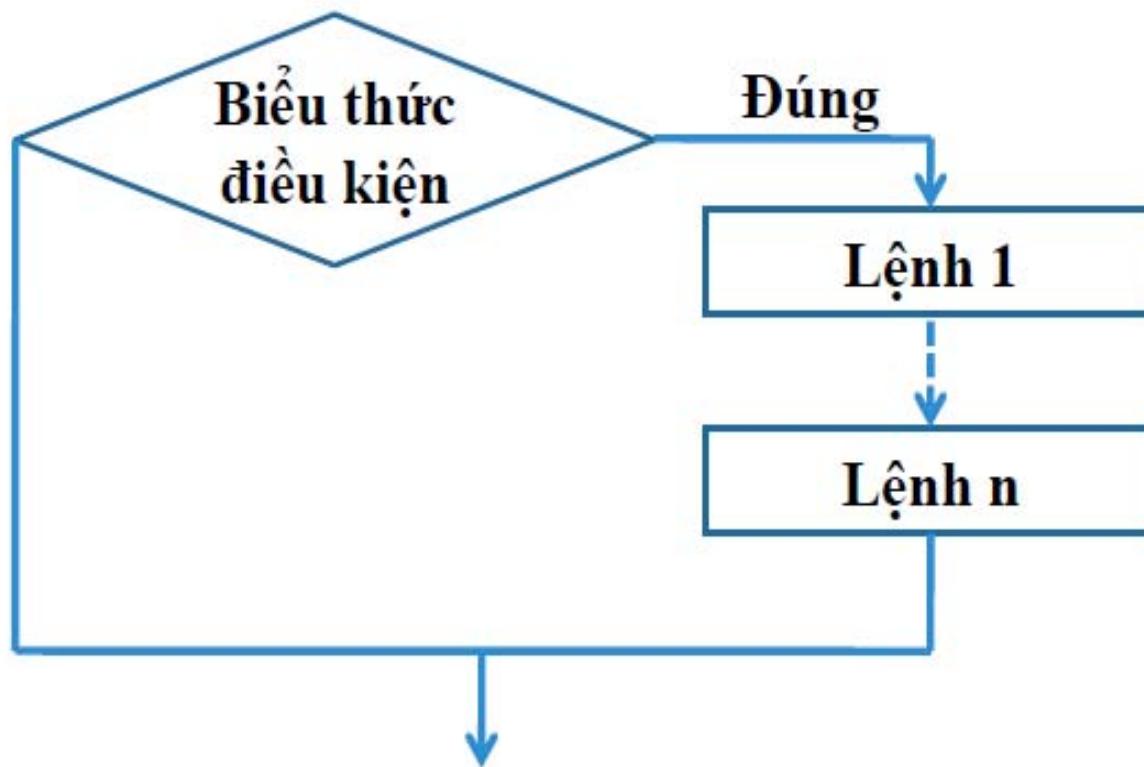
CẤU TRÚC ĐIỀU KHIỂN

- Cấu trúc điều khiển xác định thứ tự các phát biểu được thực thi.
- Cấu trúc *điều kiện và rẽ nhánh* (*if, switch, goto*) biểu diễn các “quyết định”
- Cấu trúc “*lặp*” (*for, while, do ... while*) cho phép lặp lại nhiều lần việc thực thi các phát biểu
- Cấu trúc “*ngắt*” (*break, continue, stop*)



Câu lệnh điều kiện IF

- ❖ Câu lệnh IF thiếu (chỉ xét trường hợp ĐÚNG)



Câu lệnh điều kiện IF (tt)

❖ Cú pháp:

```
if(biểu thức điều kiện)  
<câu lệnh>;
```

```
if(biểu thức điều kiện)  
{  
    <khối lệnh>;  
}
```

❖ Ví dụ 1:

```
void main()  
{  
    int x, y;  
    cout<<"Nhập giá trị cho x và y";  
    cin>>x>>y;  
    if(x >= y)  
        cout<<"Giá trị của x lớn hơn y";  
}
```

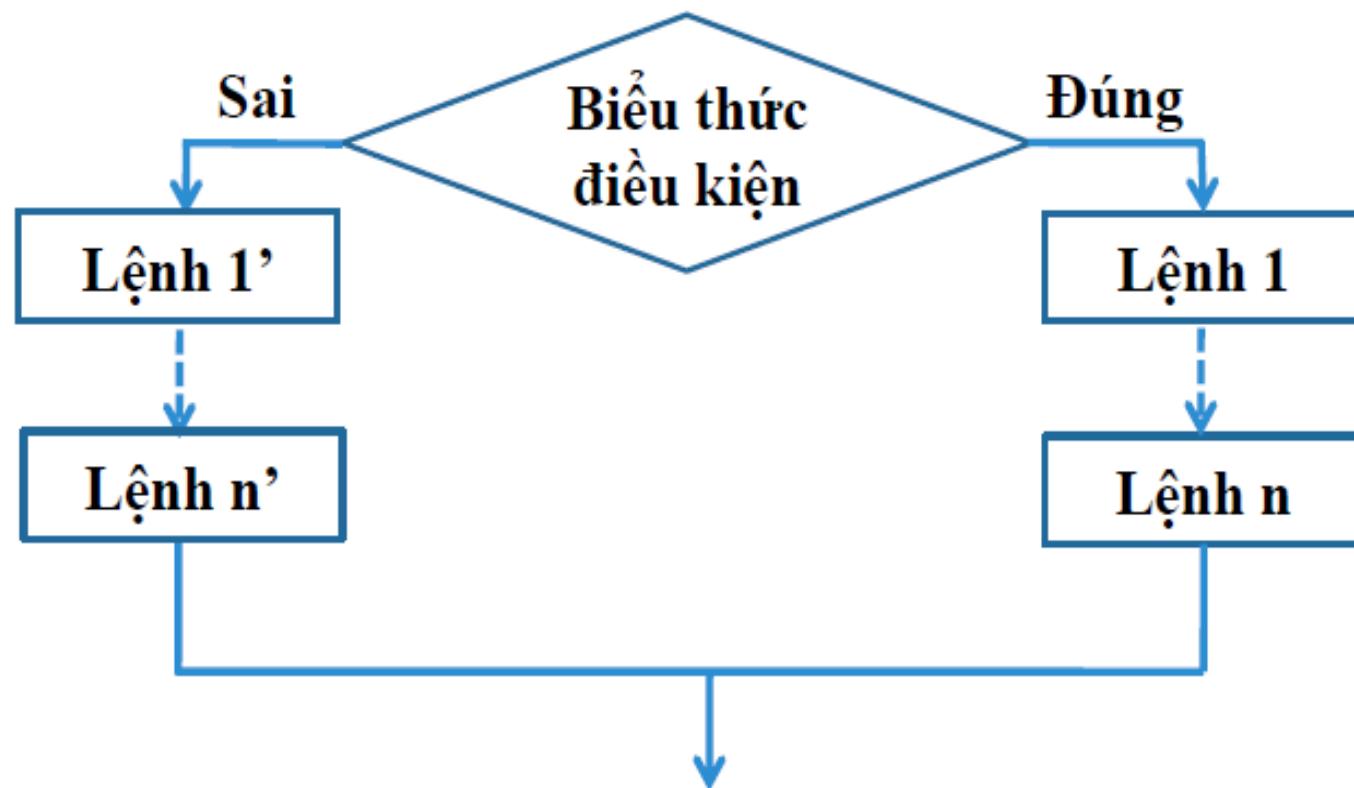
Câu lệnh điều kiện IF (tt)

❖ Ví dụ 2:

```
void main()
{
    int x, y;
    cout<<"Nhập giá trị cho x và y";
    cin>>x>>y;
    if(x >= y)
    {
        cout<<"Giá trị của x lớn hơn y";
        x = x - y;
        cout<<"Giá trị x mới: "<<x;
    }
}
```

Câu lệnh điều kiện IF (tt)

- ❖ Câu lệnh IF đủ (xét trường hợp ĐÚNG và SAI)



Câu lệnh điều kiện IF (tt)

❖ Cú pháp:

```
if(biểu thức điều kiện)
    <câu lệnh 1>;
else
    <câu lệnh 2>;
```

```
if(biểu thức điều kiện)
{
    <khối lệnh 1>;
}
else
{
    <khối lệnh 2>;
}
```

Câu lệnh điều kiện IF (tt)

❖ Ví dụ 1:

```
void main()
{
    int x, y;
    cout<<"Nhap gia tri cho x va y";
    cin>>x>>y;
    if(x >= y)
        cout<<"Gia tri cua x lon hon y";
    else //x < y
        cout<<"Gia tri cua x nho hon y";
}
```

Câu lệnh điều kiện IF (tt)

❖ Ví dụ 2:

```
void main()
{
    int x, y;
    cout<<"Nhap gia tri cho x va y";
    cin>>x>>y;
    if(x >= y)
    {
        cout<<"Gia tri cua x lon hon y";
        x = x - y;
        cout<<"Gia tri x moi: "<<x;
    }
    else      //x < y
    {
        cout<<"Gia tri cua y lon hon x";
        y = y - x;
        cout<<"Gia tri y moi: "<<y;
    }
}
```

Câu lệnh điều kiện IF (tt)

Một số lưu ý:

- Không được thêm ; sau điều kiện của if.
- Câu lệnh if có thể lồng nhau và else sẽ tương ứng với if gần nó nhất. Ví dụ:

```
if (a != 0)
    if (b > 0)
        cout<<"a != 0 va b > 0";
else
    cout<<"a != 0 va b <= 0";
    if (a !=0)
    {
        if (b > 0)
            cout<<"a != 0 va b > 0";
        else
            cout<<"a != 0 va b <= 0";
    }
```

Câu lệnh điều kiện IF (tt)

- Nên dùng else để loại trừ trường hợp.

```
if (delta < 0)
    cout<<"PT vo nghiem";
if (delta == 0)
    cout<<"PT co nghiem kep";
if (delta > 0)
    cout<<"PT co 2 nghiem phan biet";
```

```
if (delta < 0)
    cout<<"PT vo nghiem";
else //delta >= 0
{
    if (delta == 0)
        cout<<"PT co nghiem kep";
    else //delta > 0
        cout<<"PT co 2 nghiem phan biet";
}
```

Câu lệnh điều kiện IF (tt)

- Khi có nhiều hơn 2 điều kiện

if (Biểu thức điều kiện 1)

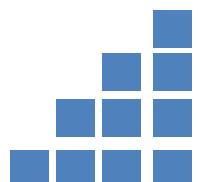
 < Câu lệnh 1>;

else if (Biểu thức điều kiện 2)

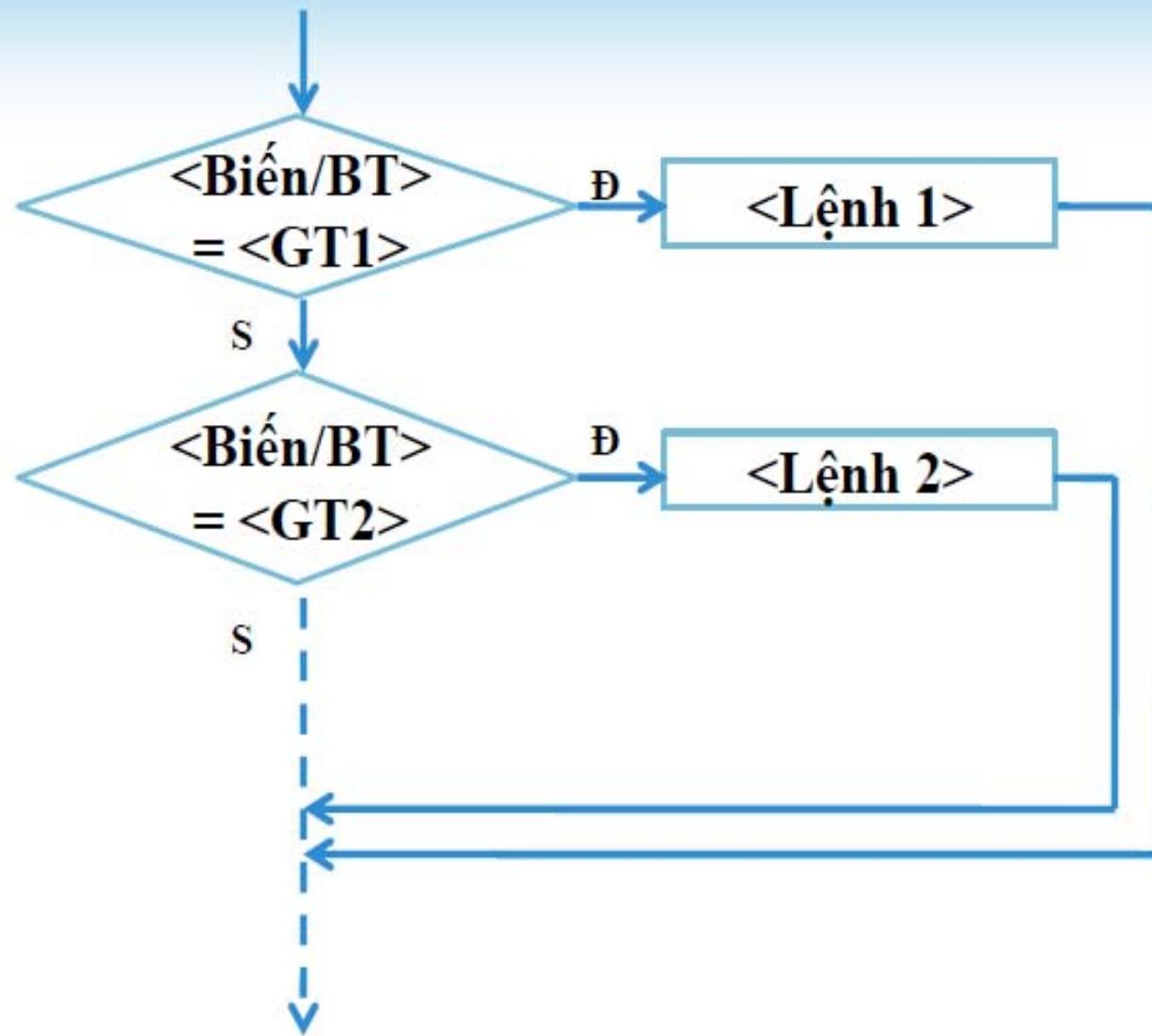
 < Câu lệnh 2>;

else

 < Câu lệnh 3>;



Câu lệnh SWITCH



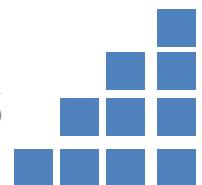
Câu lệnh SWITCH (tt)

❖ Cú pháp:

```
switch (biểu thức)
{
    case n1:
        <khối lệnh>;
        break;
    case n2:
        <khối lệnh>;
        break;
    .....
    case nk:
        <khối lệnh>;
        break;
    [default:   <khối lệnh>; ]
}
```

Câu lệnh SWITCH (tt)

- Các giá trị n_1, n_2, n_k là các hằng nguyên, hoặc ký tự.
- Các giá trị n_1, n_2, n_k không trùng nhau.
- **break**: kết thúc thực thi câu lệnh và thoát khỏi switch.
- switch sẽ thực hiện đến case tương ứng đến khi gặp break hoặc cuối switch sẽ kết thúc.
- Nếu giá trị biểu thức không thuộc các giá trị trên thì sẽ thực hiện lệnh sau từ khóa default.



Câu lệnh SWITCH (tt)

❖ Ví dụ 1:

```
void main()
{
    int n;
    cout<<"Nhập giá trị cho n (1<=n<=3)";
    cin>>n;
    switch(n)
    {
        case 1:
            cout<<"Một";
            break;
        case 2:
            cout<<"Hai";
            break;
        default:
            cout<<"Ba";
    }
}
```

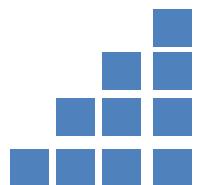
CẤU TRÚC NHẢY GOTO

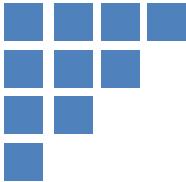
a. Ý nghĩa:

- Là một dạng khác của cấu trúc rẽ nhánh .
- Cho phép chương trình chuyển đến thực hiện một đoạn lệnh khác bắt đầu từ một điểm được đánh dấu bởi một nhãn trong chương trình.
- Lệnh goto thường được sử dụng để tạo vòng lặp.

b. Cú pháp

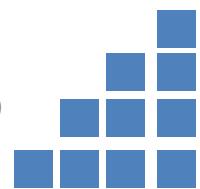
Goto <nhãn> ;





- **Ví dụ:**

```
#include <iostream>
using namespace std;
int main ()
{
    int n=10;
    loop: ;
    cout << n << ", ";
    n--;
    if (n>0) goto loop;
    cout << "Kết thúc!";
    return 0;
}
```



Trường ĐH Khoa Học Tự Nhiên
Khoa Vật Lý – Bộ Môn Vật Lý Ứng Dụng

KỸ THUẬT LẬP TRÌNH C++

Chương 2: CẤU TRÚC ĐIỀU KHIỂN

Phần 2: Cấu trúc lặp

ĐẶT VĂN ĐỀ

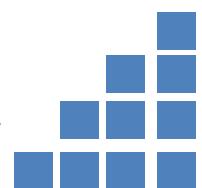
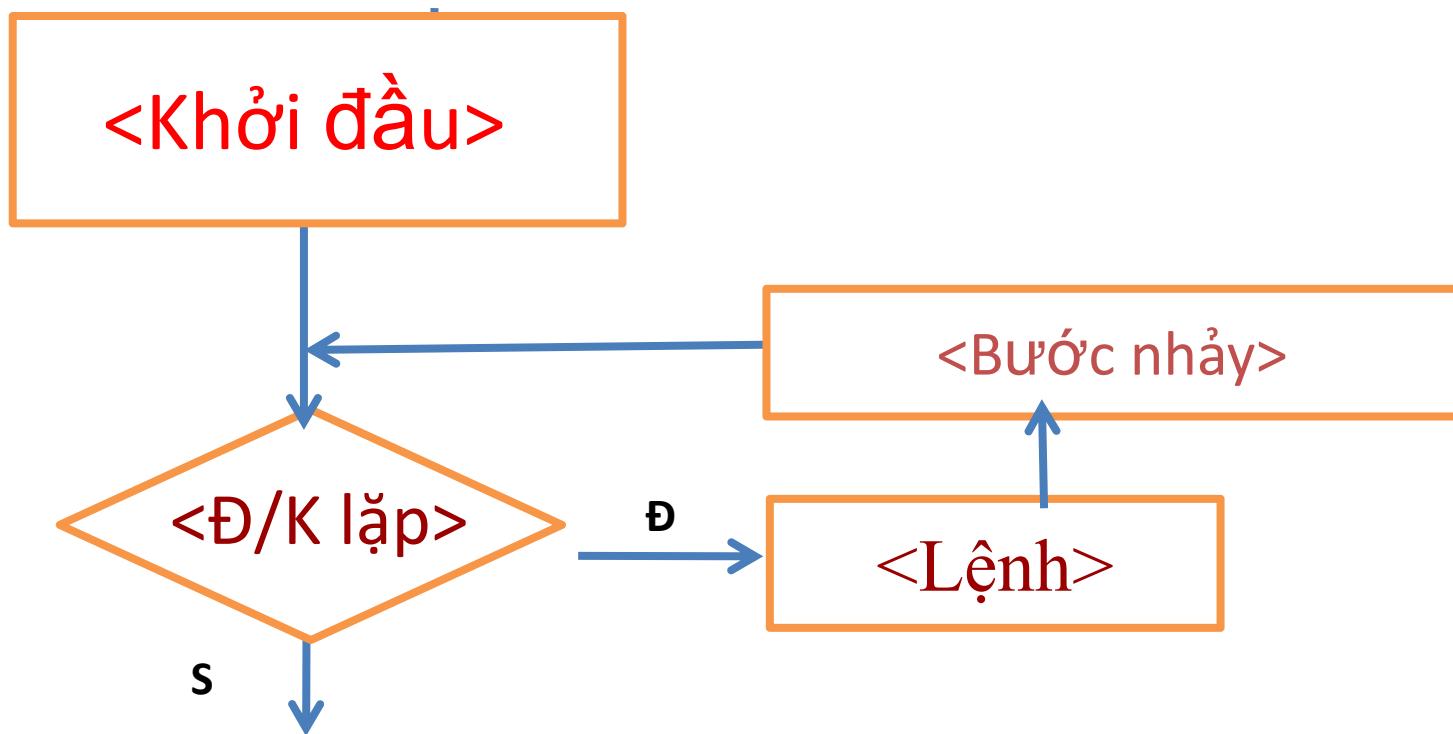
❖ Ví dụ

- Viết chương trình xuất các số từ 1 đến 10
⇒ Sử dụng 10 câu lệnh cout
- Viết chương trình xuất các số từ 1 đến 1000
⇒ Sử dụng 1000 câu lệnh cout !!!

❖ Giải pháp

- Sử dụng cấu trúc lặp để lặp lại một hành động trong khi còn thỏa một điều kiện nào đó.
- Có 3 câu lệnh lặp: for, while, do... while

Câu lệnh FOR



Câu lệnh FOR

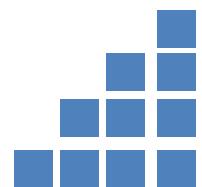
❖ Cú pháp:

for (khởi đầu ; điều kiện lặp ; bước nhảy)

{

Các câu lệnh;

}



Câu lệnh FOR

Ví dụ: Xuất câu “Hello” 10 lần, mỗi lần trên 1 dòng

```
void main()
{
    int buoc;
    for(buoc=1 ; buoc<=10 ; buoc++)
        cout<<"Hello"<<endl;
}
```

Biểu thức
khởi đầu

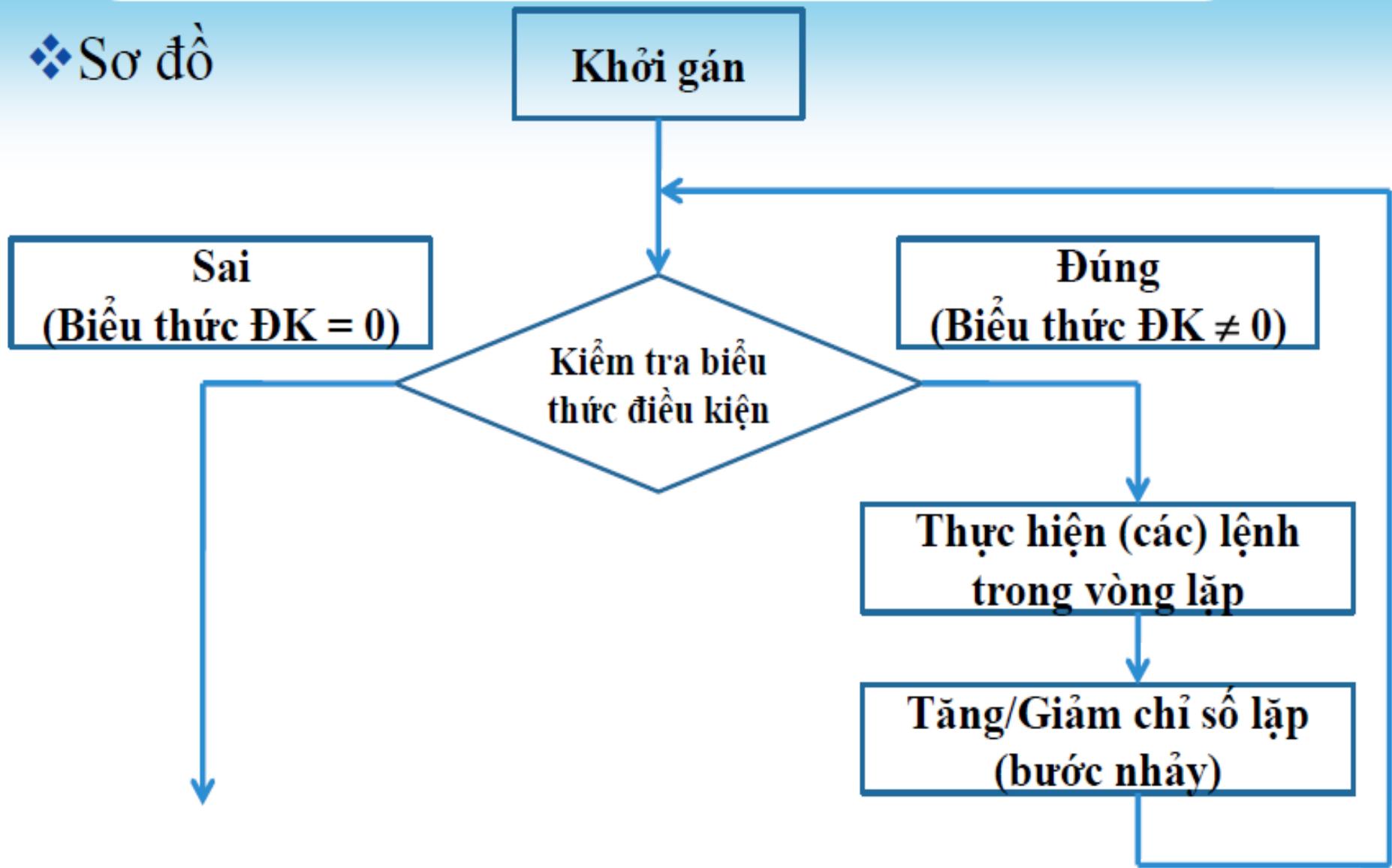
Biểu thức
điều kiện

Bước
nhảy

Bước	buoc	Đk	Xuất
1	1	buoc ≤ 10?	Hello
2	2	buoc ≤ 10?	Hello
3	3	buoc ≤ 10?	Hello
4	4	buoc ≤ 10?	Hello
5	5	buoc ≤ 10?	Hello
6	6	buoc ≤ 10?	Hello
7	7	buoc ≤ 10?	Hello
8	8	buoc ≤ 10?	Hello
9	9	buoc ≤ 10?	Hello
10	10	buoc ≤ 10?	Hello
11		buoc ≤ 10?	

Câu lệnh FOR (tt)

❖ Sơ đồ



Câu lệnh FOR (tt)

- ❖ Ví dụ 2: Tính tổng $S(n) = 1+2+3+\dots+n$, $n > 0$

```
void main()
{
    int s=0, i;
    cout<<"Nhập n: ";
    cin>>n;
    for(i=1 ; i<=n ; i++)
        s=s+i;
    cout<<"Tổng = "<<s;
}
```

Ví dụ: $n = 5$

Câu Lệnh WHILE

❖ Cú pháp

```
<Biểu thức khởi đầu>;  
while(<Biểu thức điều kiện>)  
{  
    <Khối lệnh>;  
    <Bước nhảy>;  
}
```

❖ Ví dụ 1: Xuất câu “Hello!” 10 lần, mỗi lần trên 1 dòng

Phát biểu while (tt)

Cách thực thi: Khi **while** thực thi :

- B1.** Tính toán biểu thức **<biểu thức đk>**.
- B2.** Nếu kết quả của **<biểu thức đk>**. là **TRUE ($\neq 0$)**, thì sang **B3**.
Nếu kết quả của **<biểu thức đk>**. là **FALSE ($= 0$)**, thì sang **B4**.
- B3.** Thực thi **câu lệnh 1, câu lệnh 2,...** thân của **while**.
Quay trở về **B1**.
- B4.** Kết thúc **while**.

(các phát biểu sau **while** tiếp tục thực thi).

Phát biểu while (tt)

Đặc điểm

- Không được thêm ; ngay sau lệnh lệnh while
- Khối lệnh lặp có thể không được thực hiện lần nào nếu điều kiện sai ngay từ đầu.
- Để vòng lặp không lặp vô hạn thì trong khối lệnh thông thường phải có ít nhất một câu lệnh nào đó gây ảnh hưởng đến kết quả của điều kiện, ví dụ làm cho điều kiện đang đúng trở thành sai.
- Nếu điều kiện luôn luôn nhận giá trị đúng (ví dụ biểu thức điều kiện là 1) thì trong khối lệnh lặp phải có câu lệnh kiểm tra dừng và lệnh break.

Câu lệnh WHILE (tt)

```
void main()
{
    int buoc;
    buoc = 1;
    while (buoc <= 10)
    {
        cout<<"Hello"<<endl;
        buoc++;
    }
}
```

Biểu thức khởi đầu

Biểu thức điều kiện

Bước nhảy

Câu lệnh WHILE (tt)

- ❖ Ví dụ 2: Tính tích $S(n) = 1*2*3*...*n$, $n > 0$

```
void main()
{
    int i;
    long s =
        i = 1;
    while(i<=n)
    {
        s=s*i;
        i++;
    }
    cout<<"Tich"
}
```

Ví dụ: $n = 5$

Câu lệnh DO WHILE

❖ Cú pháp

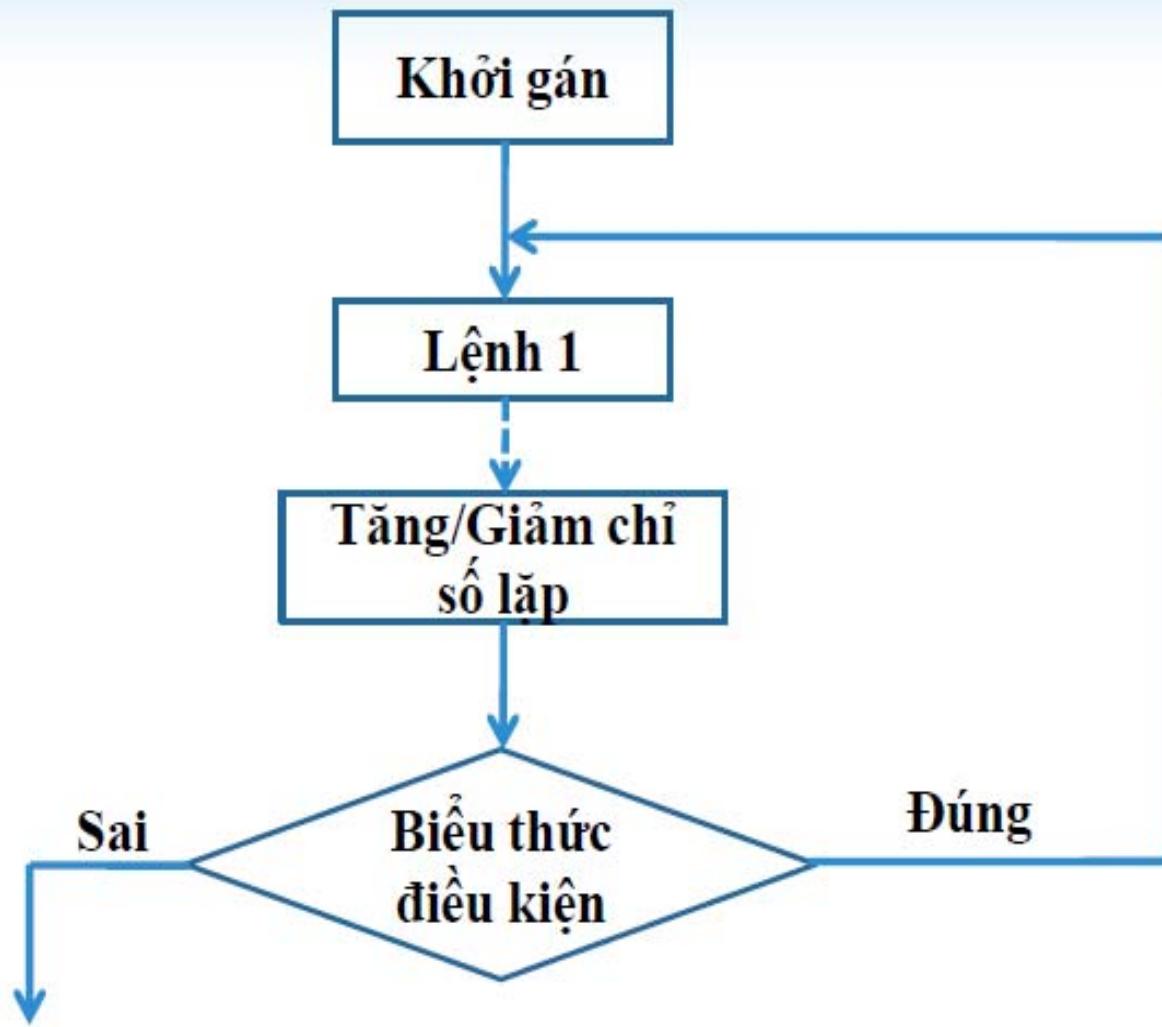
```
<Biểu thức khởi đầu>;  
do  
{  
    <Khối lệnh>;  
    <Bước nhảy>;  
} while(<Biểu thức điều kiện>);
```

Lệnh lặp do ... while

- Vòng *do....while* kiểm tra điều kiện tại điểm cuối của vòng lặp, khác với *for* hay *while* – kiểm tra điều kiện ngay đầu vòng lặp. Phát biểu **do-while** tính và xét biểu thức *sau khi* thực thi các phát biểu trong phần thân

Câu lệnh DO...WHILE (tt)

❖ Sơ đồ



Câu lệnh DO...WHILE (tt)

- Ví dụ : Xuất câu “Hello!” 10 lần, mỗi lần trên 1 dòng

```
void main()
{
    int buoc;
    buoc = 1; ——————
    do
    {
        cout<<"Hello"<<endl;
        buoc++; ——————
    } while(buoc<10); ——————
}
```

Biểu thức khởi đầu

Bước nhảy

Biểu thức điều kiện

Câu lệnh DO... WHILE (tt)

- Câu lệnh DO... WHILE sẽ được thực hiện ít nhất 1 lần

```
void main()
{
    int n;
    do
    {
        cout<<"Nhập n: ";
        cin>>n;
    }while (n < 1 || n > 100);
}
```

⇒ Thường sử dụng trong trường hợp nhập có kiểm tra.

So sánh for, while, do While

- Đều có thể lặp lại nhiều hành động.

Ví dụ: $n = 10$;

```
for (int i = 1; i <= n; i++)
    cout<<i<<endl;
```

```
int i = 1;
while (i <= n)
{
    cout<<i<<endl;
    i++;
}
```

```
int i = 1;
do
{
    cout<<i<<endl;
    i++;
} while (i <= n);
```

So sánh (tt)

- Số lần lặp thường được xác định ngay trong câu lệnh FOR, trong khi WHILE và DO...WHILE có thể sử dụng điều kiện khác để thoát.

```
int n = 10;  
for (int i = 1; i <= n; i++)  
    ...;
```

```
int i = 1;  
while (1)  
{  
    ...;  
}
```

```
int i = 1;  
do  
{  
    ...;  
} while (1);
```

So sánh (tt)

- FOR, WHILE có thể không thực hiện lần nào.
- DO... WHILE sẽ được thực hiện ít nhất 1 lần.

```
int n = 100;  
while (n < 10)  
{  
    cout<<n<<"\t";  
    n++;  
}
```

```
int n = 100;  
do  
{  
    cout<<n<<"\t";  
    n++;  
}while (n < 10);
```

MỘT SỐ LUU Y

- Nếu có nhiều thành phần trong từng biểu thức của FOR, thì các thành phần cách nhau bằng dấu ,

Ví dụ:

```
int i, j;  
for (i=1, j=2 ; i+j < 10 ; i++, j+=2)  
    cout<<(i + j);
```

Một số lưu ý (tt)

- Nếu có nhiều thành phần trong biểu thức điều kiện của WHILE và DO...WHILE, thì các điều kiện kết hợp với nhau bằng phép AND (`&&`) hoặc phép OR (`||`)

Ví dụ:

```
int i=1, j=1, n = 3, m = 5;
while(i<=n && j<m)
{
    cout<<i+j<<"\t";
    i++;
    j++;
}
```

Một số lưu ý (tt)

- Có thể có các câu lệnh lặp lồng nhau. Ví dụ:

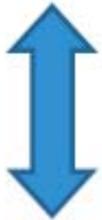
```
int i, j, n = 3, m = 5;  
for (i = 1; i <= n; i++)  
{  
    for (j = 1; j <= m; j++)  
    {  
        cout<<(i + j);  
        cout<<"\n";  
    }  
}
```

Một số lưu ý (tt)

- Trong câu lệnh FOR có thể không có <Biểu thức khởi đầu>

Ví dụ:

```
int i;  
for (i = 1 ; i <= 10 ; i++)  
    cout<<i<<endl;
```



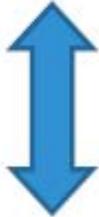
```
int i = 1;  
for ( ; i <= 10 ; i++)  
    cout<<i<<endl;
```

Một số lưu ý (tt)

- Trong câu lệnh lặp có thể không có <Bước nhảy>

Ví dụ:

```
int i;  
for (i = 1 ; i <= 10 ; i++ )  
    cout<<i<<endl;
```



```
int i;  
for (i = 1 ; i <= 10 ; )  
{  
    cout<<i<<endl;  
    i++;  
}
```

Một số lưu ý (tt)

- Trong câu lệnh lặp có thể không có <Biểu thức điều kiện>

Ví dụ:

```
int i;  
for (i = 1; i <= 10; i++ )  
    cout<<i<<endl;
```



```
int i;  
for (i = 1; ; i++ )  
{  
    if(i > 10)  
        break;  
    cout<<i<<endl;  
}
```

Một số lưu ý (tt)

- Lệnh break làm kết thúc vòng lặp
- Lệnh continue bỏ qua lần lặp hiện tại

Ví dụ:

```
for (i = 1; i <= 10; i++)
{
    if (i % 2 == 0)
        break;
    cout<<i;
}

for (i = 1; i <= 10; i++)
{
    if (i % 2 == 0)
        continue;
    cout<<i;
}
```

Một số lưu ý (tt)

- Vòng lặp đi xuôi: giá trị bước nhảy tăng dần
- Vòng lặp đi ngược: giá trị bước nhảy giảm dần

Ví dụ:

```
int i;  
for (i = 1 ; i <= 10 ; i++)  
    cout<<i<<"\t";
```

⇒ Xuất:

```
int i;  
for (i = 10 ; i >= 1 ; i--)  
    cout<<i<<"\t";
```

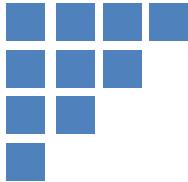
⇒ Xuất:

Một số lưu ý (tt)

- Câu lệnh lặp có thể không được thực hiện lần nào, nếu ngay từ lần đầu điều kiện lặp đã không thỏa. Ví dụ:

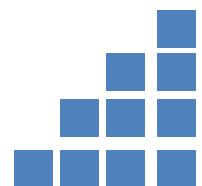
```
int i, n=0;  
for (i = 1 ; i <= n ; i++)  
    cout<<i<<"\t";
```

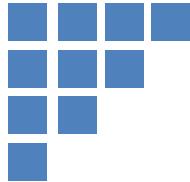
```
int n = 1;  
while (n > 10)  
{  
    cout<<n<<endl;  
    n--;  
}
```



Phát biểu break

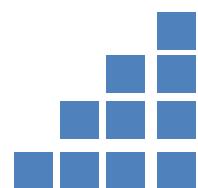
- Phát biểu **break** kết thúc cấu trúc lặp gần nhất mà nó xuất hiện trong đó.
- Kết thúc cấu trúc lặp mà *không cần biết kết quả của <điều kiện lặp>*
- Được dùng trong trường hợp thoát khỏi vòng lặp mà không phải kiểm tra tất cả mọi trường hợp.
- Phát biểu **break** thường xuất hiện cùng với phát biểu **if**.

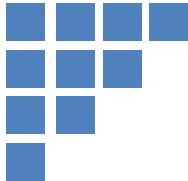




Lệnh continue

Lệnh dùng để quay lại đầu vòng lặp mà không chờ thực hiện hết các lệnh trong khối lệnh lặp.





BÀI TẬP ÁP DỤNG

1. viết chương trình nhập ba số a, b, c. Và tìm số lớn nhất trong ba số đó.
2. Nhập vào hai số a,b. Khảo sát mối tương quan của hai số đó.
3. Nhập vào 3 số nguyên. Kiểm tra xem đó có phải là ba cạnh của tam giác không. Nếu có đó là tam giác gì?
4. Viết chương trình nhập vào 4 số nguyên. Tìm và in ra số lớn nhất và nhỏ nhất trong 4 số đó.



5. Viết chương trình cho phép nhập vào 1 ký tự. Cho biết đó là chữ hoa, chữ thường hay không phải các loại trên.
6. Nhập 4 số nguyên a, b, c và d. Hãy sắp xếp giá trị của 4 số nguyên này theo thứ tự tăng dần.
7. Giải hệ phương trình bậc nhất hai ẩn $\{ax+by=c; a'x+b'y=c'\}$
8. Kiểm tra 1 số có là số nguyên tố không?
9. Tìm ước số chung lớn nhất của 2 số nguyên dương a và b nhập từ bàn phím

10. Viết chương trình in các ký tự từ ‘A’ đến ‘Z’ xuôi và ngược, chữ hoa và chữ thường.

11. Viết chương trình in tam giác Pascal.

12. Viết chương trình cho phép nhập vào chiều cao. In ra tam giác vuông cân đặc có chiều cao tương ứng.

Ví dụ: Chiều cao 4

*

* * *

* * * * *

* * * * * *

13. Viết chương trình cho phép nhập vào chiều cao. In ra tam giác vuông cân rỗng có chiều cao tương ứng.

Ví dụ: Chiều cao 4

*

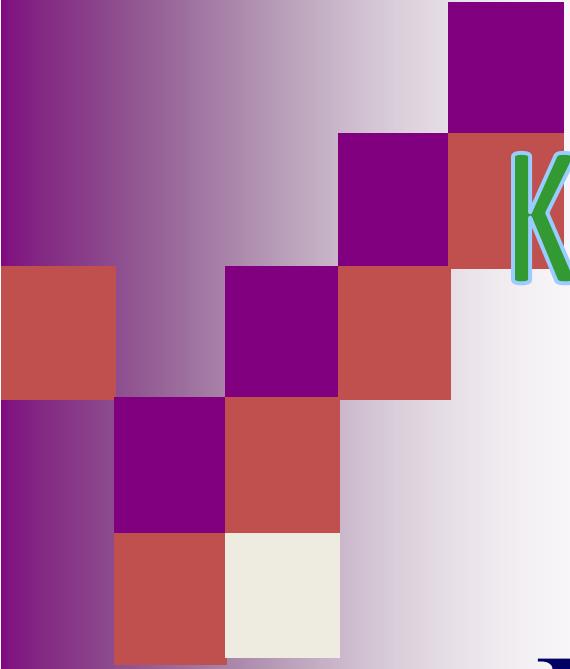
* * *

* * *

* * * * * * *

14. Viết chương trình cho phép nhập vào chiều rộng và chiều cao. In ra hình chữ nhật đặc có chiều rộng và chiều cao tương ứng.
15. Viết chương trình cho phép nhập vào chiều rộng và chiều cao. In ra hình chữ nhật rỗng có chiều rộng và chiều cao tương ứng.

Trường ĐH Khoa Học Tự Nhiên
Khoa Vật Lý – Bộ Môn Vật Lý Ứng Dụng



KỸ THUẬT LẬP TRÌNH C++

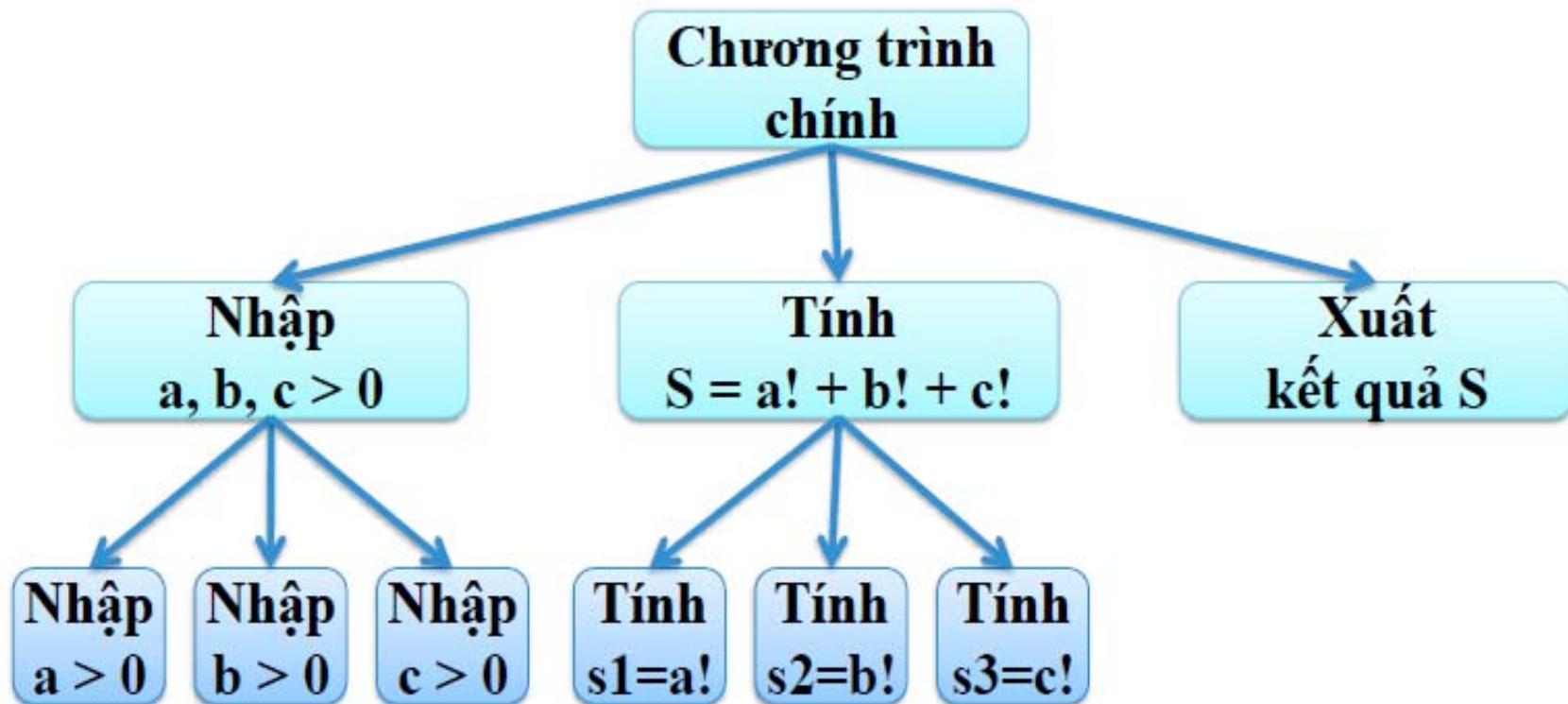
Chương 3:

FUNCTION

Đặt vấn đề

❖ Ví dụ

- Viết chương trình tính $S = a! + b! + c!$, với a, b, c là ba số nguyên dương được nhập từ bàn phím



Đặt vấn đề (tt)

- ❖ 3 đoạn lệnh nhập a, b, c > 0

```
do {  
    cout<<"Nhap mot so nguyen duong: ";  
    cin>>a;  
} while (a <= 0);  
  
do {  
    cout<<"Nhap mot so nguyen duong: ";  
    cin>>b;  
} while (b <= 0);  
  
do {  
    cout<<"Nhap mot so nguyen duong: ";  
    cin>>c;  
} while (c <= 0);
```

Đặt vấn đề (tt)

❖ Giải pháp => Viết 1 lần và sử dụng nhiều lần

- Đoạn lệnh nhập tổng quát, với n = a, b, c

```
do {  
    cout<<"Nhap mot so nguyen duong: ";  
    cin>>n;  
} while (n <= 0);
```

- Đoạn lệnh tính giai thừa tổng quát, với n = a, b, c

```
//Tinh s = n! = 1 * 2 * ... * n  
s = 1;  
for (i = 2; i <= n ; i++)  
    s = s * i;
```

HÀM

❖ Khái niệm

- Hàm là một đoạn chương trình độc lập, thực hiện trọn vẹn một công việc nhất định, sau đó trả về giá trị cho chương trình gọi nó. Nói cách khác, hàm là một sự chia nhỏ chương trình.

❖ Mục đích sử dụng hàm

- Khi có một công việc giống nhau cần thực hiện ở nhiều vị trí,
- Khi cần chia một chương trình lớn phức tạp thành các đơn thể nhỏ để chương trình được trong sáng, dễ hiểu, dễ quản lý.

HÀM (tt)

❖ Cú pháp

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

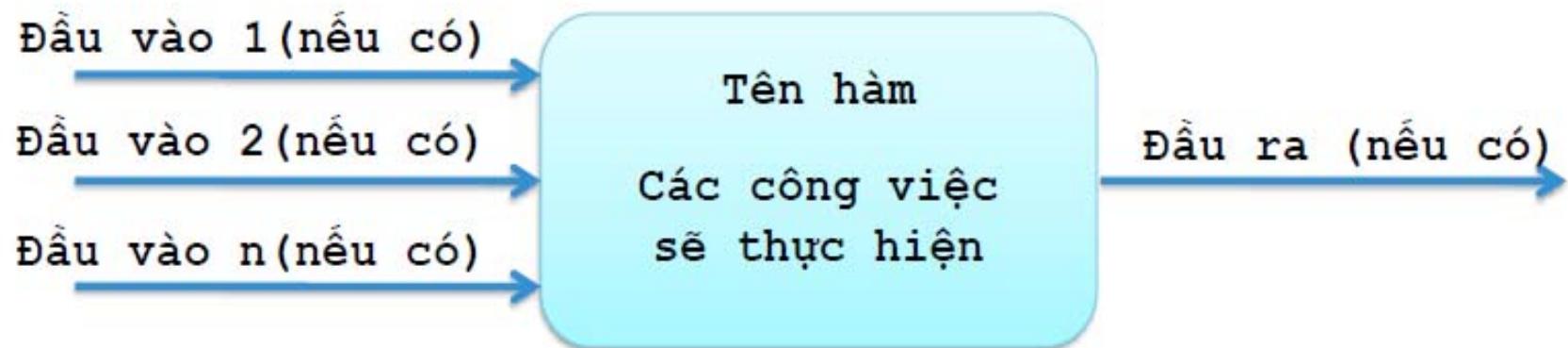
Trong đó:

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

CÁC BƯỚC XÂY DỰNG HÀM

❖ Cần xác định các thông tin sau đây:

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



CÁC VÍ DỤ

❖ Ví dụ 1

- Tên hàm: TinhTong
- Công việc: tính và xuất tổng 2 số nguyên
- Đầu vào: hai số nguyên x và y
- Đầu ra: không có

```
void TinhTong(int x, int y)
{
    int s;
    s = x + y;
    cout<<"Tong "<<x<<" + "<<y<<" = "<<s;
}
```

CÁC VÍ DỤ (tt)

❖ Ví dụ 2

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

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

TÂM VỰC

❖ Khái niệm

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

TÂM VỰC

```
int a;  
void Ham1()  
{  
    int a1;  
}
```

```
int Ham2()  
{  
    int a2;  
    [  
        int a21;  
    ]  
}
```

```
void main()  
{  
    int a3;  
}
```

LUU Y

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

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

void main()
{
    ...
}

void XuatTong(int x, int y)
{
    cout<<x<<" + "<<y<<" = "<<(x + y);
}
```

- Để một hàm thực thi, cần **gọi hàm** với tên hàm và chuyển các đối số cho hàm.
- Các đối số phải tương ứng với danh sách tham số hình thức cả về kiểu và thứ tự.
- Có hai hình thức chuyển đổi số cho hàm:
 - Truyền tham trị
 - Truyền tham chiếu

CÁC CÁCH TRUYỀN ĐỐI SỐ

❖ Truyền Giá trị (Call by Value)

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

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

```
1. #include <iostream>
2. using namespace std;
3.
4. int sum(int a, int b)
5. {
6.     return (a+b);
7. }
8.
9. int main()
10. {
11.     int a = 2, b = 3;
12.     int s;
13.     s = sum(a, b); // Gán giá trị là tổng a và b cho biến s;
14.     cout << "Tổng của a và b: " << s << endl;
15.     return 0;
16. }
```

```
1. #include <iostream>
2. using namespace std;
3. void swap(int a, int b)
4. {
5.     int tamp = a;
6.     a = b;
7.     b = tamp;
8. }
9.
10. int main()
11. {
12.     int a = 2, b = 3;
13.     cout << "Truoc khi hoan doi: " << a << ' ' << b << endl;
14.     swap(a, b); // hoán đổi
15.     cout << "Sau khi hoan doi: " << a << ' ' << b << endl;
16.     return 0;
17. }
```

CÁC CÁCH TRUYỀN ĐỐI SỐ

❖ Truyền Địa chỉ (Call by Address)

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

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

CÁC CÁCH TRUYỀN ĐỐI SỐ

❖ Truyền Tham chiếu (Call by Reference)

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

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

```
1. #include <iostream>
2. using namespace std;
3. void swap(int &a, int &b)
4. {
5.     int tamp = a;
6.     a = b;
7.     b = tamp;
8. }
9.
10. int main()
11. {
12.     int a = 2, b = 3;
13.     cout << "Truoc khi hoan doi: " << a << ' ' << b << endl;
14.     swap(a, b); // hoán đổi
15.     cout << "Sau khi hoan doi: " << a << ' ' << b << endl;
16.     return 0;
17. }
```

LƯU Ý KHI TRUYỀN ĐỐI SỐ

- Trong một hàm, các tham số có thể truyền theo nhiều cách.

```
void HonHop(int x, int &y)
{
    ...
    x++;
    y++;
}
```

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

LUU Y KHI TRUYEN DOI SO

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

void TinhTong(int x, int y, int &tong)
{
    tong = x + y;
}

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

LỜI GỌI HÀM

❖ Cách thực hiện

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

Đối với hàm có kiểu void:

<tên hàm> (<đối số 1>, ..., <đối số n>);

Đối với hàm có kiểu khác void

<tên biến> = <tên hàm> (<đối số 1>, ..., <đối số n>);

LỜI GỌI HÀM

❖ Ví dụ

```
//Cac ham duoc khai bao va xay dung o day
void main()
{
    int n = 9;
    XuatTong(1, 2);
    XuatTong(1, n);
    TinhTong(1, 2);
    int tong = TinhTong(1, 2);
    TruyenGiaTri(1);
    TruyenGiaTri(n);
    TruyenDiaChi(1);
    TruyenDiaChi(&n);
    TruyenThamChieu(1);
    TruyenThamChieu(n);
}
```

LỜI GỌI CHƯƠNG TRÌNH CON

❖ Ví dụ

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

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

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

Phát biểu return

- Phát biểu **return** nhằm dừng thực thi hàm, trả về nơi gọi nó;
- Được dùng để trả giá trị (tính toán được) về cho nơi gọi hàm.
- Trong một hàm, có thể có nhiều phát biểu return, nhưng chỉ 1 phát biểu return được thực thi,
- Một phát biểu return chỉ có thể có tối đa 1 giá trị được trả về.

HÀM ĐỆ QUY

❖ Khái niệm

- Một chương trình con có thể gọi một chương trình con khác.
- Nếu gọi chính nó thì được gọi là sự đệ quy.
- Số lần gọi này phải có giới hạn (điểm dừng)

❖ Ví dụ

- Tính $S(n) = n! = 1 * 2 * \dots * (n-1) * n$
- Ta thấy $S(n) = S(n-1) * n$
- Vậy thay vì tính $S(n)$ ta sẽ đi tính $S(n-1)$
- Tương tự tính $S(n-2), \dots, S(2), S(1), S(0) = 1$

HÀM ĐỆ QUY

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

HÀM TRÙNG TÊN

- Trong cùng một chương trình, một chương trình con có thể có tên trùng với một chương trình con khác, nhưng phải khác kiểu dữ liệu VÀ khác các tham số
- Ví dụ
 - float TBChan(int a[], int n);
 - void TBChan(int a[], int n, int x);

TỔNG KẾT

❖ Nguyên tắc xây dựng hàm

Trước khi xây dựng hàm, cần trả lời các câu hỏi:

- Hàm làm công việc gì? → Xác định tên hàm
- Hàm trả về giá trị gì? → Xác định kiểu dữ liệu hàm
- Cần những thông tin gì để hàm xử lý? → Xác định danh sách các tham số
- Ứng với mỗi thông tin cần thiết, xác định xem đã có thông tin nào có giá trị trước khi hàm xử lý chưa?
 - ✓ Nếu chưa có, hoặc đã có giá trị mà giá trị này sẽ thay đổi sau khi hàm xử lý → truyền tham chiếu
 - ✓ Nếu đã có giá trị mà giá trị này không cần thay đổi sau khi hàm xử lý → truyền tham trị

TỔNG KẾT

Hàm <u>void</u> không tham số	Hàm <u>void</u> có tham số	Hàm <u>khác void</u> không tham số	Hàm <u>khác void</u> có tham số
<code>void TenHam() { //than ham }</code>	<code>void TenHam(int x, long y) { //than ham }</code>	<code>int TenHam() { //than ham return ???; }</code>	<code>long TenHam(int x, long y) { //than ham return ???; }</code>
Cách gọi hàm			
<code>TenHam();</code>	<code>TenHam(a,b);</code>	<code>int kq=TenHam();</code>	<code>long kq=TenHam(a,b);</code>

BÀI TẬP

1. Viết chương trình tính diện tích và chu vi của hình chữ nhật với chiều dài và chiều rộng được nhập từ bàn phím
2. Viết chương trình nhập vào 4 số nguyên và sắp xếp 4 số nguyên tăng dần.

3. Tính $F(x)$

Cho hàm $F(x)$, $x \geq 0$ được định nghĩa như sau:

$F(x) = x$, nếu $x \leq 9$, $F(x) = F(S(x))$, nếu $x > 9$

Trong đó $S(x)$: tổng các chữ số của x .

Yêu cầu: Hãy viết chương trình tính $F(n!)$, với $1 \leq n \leq 500$

Bài 4: Viết chương trình trong đó có sử dụng hàm để nhập và tính giai thừa của một số nguyên.

```
include<iostream>
using namespace std;
int factorial(int n);

int main ()
{
    int n1,fact;
    cout <<"Enter the number whose
factorial has to be calculated" << endl;
    cin >> n1;
    fact=factorial(n1);
    cout << "The factorial of " << n1 << " is :
" << fact << endl;
    return(0);
}

int factorial(int n)
{
    int i=0,fact=1;
    if(n<=1)
    {
        return(1);
    }
    else
    {
        for(i=1;i<=n;i++)
        {
            fact=fact*i;
        }
        return(fact);
    }
}
```

Trường ĐH Khoa Học Tự Nhiên
Khoa Vật Lý – Bộ Môn Vật Lý Ứng Dụng



KỸ THUẬT LẬP TRÌNH C++

Chương 4: MẢNG - CHUỖI KÝ TỰ

Mục tiêu bài học

1. Biết cách làm việc của mảng, chuỗi.
2. Biết cách sử dụng mảng.
3. Biết cách trình bày và sử dụng mảng nhiều chiều và các loại mảng khác nhau.

DỮ LIỆU KIỂU MẢNG

Mảng 1 chiều

Đặt vấn đề

❖ Ví dụ

- Chương trình cần lưu trữ 3 số nguyên?
=> Khai báo 3 biến int a1, a2, a3;
- Chương trình cần lưu trữ 100 số nguyên?
=> Khai báo 100 biến kiểu số nguyên!
- Người dùng muốn nhập n số nguyên?
=> Không thực hiện được!

❖ Giải pháp

- Kiểu dữ liệu mới cho phép lưu trữ một dãy các số nguyên.

Dữ liệu kiểu mảng

❖ Khái niệm

- Là một kiểu dữ liệu có cấu trúc do người lập trình định nghĩa.
- Biểu diễn một dãy các biến có cùng kiểu. Ví dụ: dãy các số nguyên, dãy các ký tự...
- Kích thước được xác định ngay khi khai báo.
- NNLT C luôn chỉ định một khối nhớ liên tục cho một biến kiểu mảng.

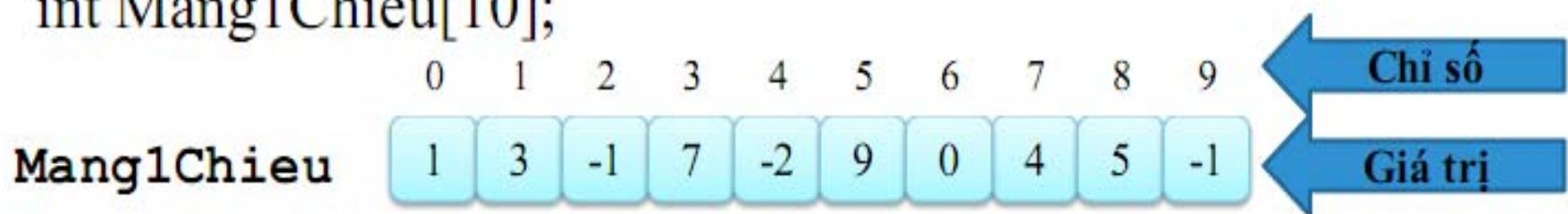
Dữ liệu kiểu mảng (tt)

❖ Khai báo

<kiểu cơ sở> <tên biến mảng>[<số phần tử>];

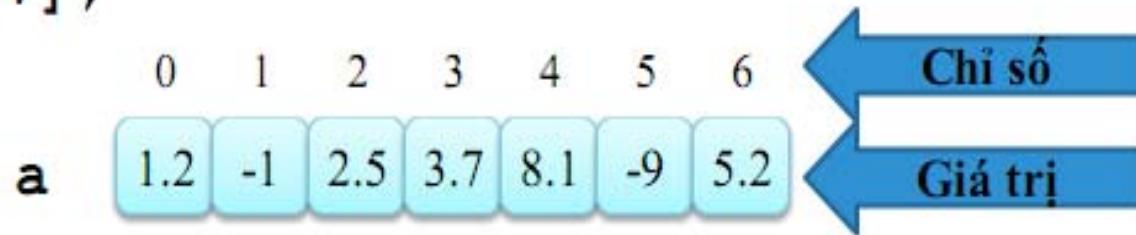
❖ Ví dụ 1

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



❖ Ví dụ 2

```
float a[7];
```



Số phần tử mảng

- ❖ Phải xác định cụ thể số phần tử ngay lúc khai báo, không được sử dụng biến chưa có giá trị

```
| int n1; int a[n1]; //sai  
| int n2 = 10; int a[n2];
```

- ❖ Nên sử dụng chỉ thị tiền xử lý #define để định nghĩa số phần tử mảng

```
| #define n3 10  
| int a[n3]; // ⇔ int a[10];
```

Khởi tạo giá trị của mảng

- Khởi gán giá trị các phần tử của mảng

```
int a[5] = {1, 2, 3, 4, 5};           // khởi gán giá trị cho các phần tử của mảng a  
double b[3] = {1, n, 2};             // sai nếu n không phải hằng double
```

- Nếu số giá trị không đủ, các phần tử còn lại sẽ có giá trị là 0

```
int a[5] = {1, 2, 3};                // khởi gán giá trị cho các phần tử của mảng a  
                                    // các phần tử thứ 4 và 5 có giá trị là 0  
double b[3] = {0};                  // khởi gán giá trị 0 cho các phần tử của mảng b
```

- Nếu số giá trị nhiều hơn số phần tử sẽ sinh ra lỗi biên dịch.

```
double b[3] = {1, 2, 3, 0};          // sinh lỗi biên dịch
```

- Nếu số phần tử trong khai báo được bỏ qua, số phần tử của mảng sẽ được xác định là số giá trị trong danh sách.

```
int a[] = {1, 2, 3};                // khởi gán giá trị cho các phần tử của mảng a  
                                    // và a là một biến mảng có 3 phần tử
```



Truy xuất các phần tử của mảng

- Không thể gán, truy xuất trực tiếp biến mảng mà phải làm lần lượt với các phần tử của mảng.
- Truy xuất phần tử của mảng dùng phép toán []

định_danh[vị_trí_phần_tử]

- vị_trí_phần_tử: là một hằng nguyên hoặc một biểu thức nguyên có giá trị nhỏ hơn số phần tử của mảng.
- Phần tử đầu tiên ứng với vị trí 0, phần tử cuối cùng ứng với vị trí (số_phần_tử - 1).
- Mỗi phần tử của mảng có thể được xem như các biến thông thường.

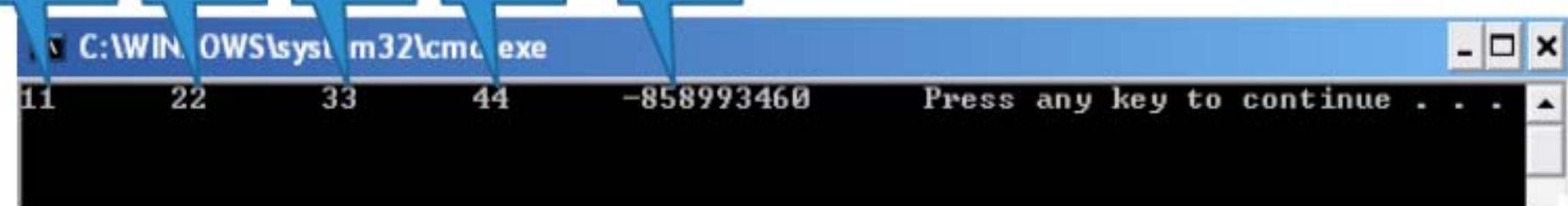
```
int a[5] = {1, 2, 3, 4, 5};  
int b = a[0]; // truy xuất phần tử đầu tiên của mảng a  
a[0] = 0; // gán giá trị cho phần tử đầu tiên của mảng a  
printf("%d", a[0]); // xuất giá trị của một phần tử của mảng a  
scanf("%d", &a[0]); // nhập dữ liệu trực tiếp cho phần tử của mảng a
```

Truy xuất đến một phần tử (tt)

```
#pragma once
#include<iostream>
using namespace std;

void main()
{
    int i, a[]={11, 22, 33, 44};
    for(i=0; i<5; i++)
        cout<<a[i]<<"\t";
}
```

i= 0 i= 1 i= 2 i= 3 i= 4



Bài tập:

1. Nhập mảng gồm n số nguyên và xuất ra mảng.
2. Tính tổng các phần tử của mảng.
3. Sắp xếp các phần tử của mảng theo chiều tăng dần.
4. Liệt kê các phần tử chẵn của mảng.
5. Đếm các phần tử của mảng là số nguyên tố.

DỮ LIỆU KIỂU MẢNG

Mảng 2 chiều

Để thuận tiện trong việc biểu diễn các loại dữ liệu phức tạp như ma trận hoặc các bảng biểu nhiều chiều, C++ đưa ra kiểu dữ liệu mảng nhiều chiều. Tuy nhiên việc sử dụng mảng nhiều chiều gấp nhiều khó khăn khi lập trình nên trong mục này chúng ta chỉ bàn đến mảng hai chiều.

Mảng 2 chiều

- Khai báo biến mảng 2 chiều

kiểu_phần_tử định_danh[số_hàng][số_cột];

- Khởi gán mảng 2 chiều

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



1	2	3
4	5	6

Các dòng cách nhau bởi dấu phẩy, mỗi dòng được bao bởi cặp ngoặc {} và toàn bộ giá trị khởi tạo nằm trong cặp dấu {}.

- Nếu số giá trị không đủ, các phần tử còn lại sẽ có giá trị là 0

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



1	0	0
4	5	6

Mảng 2 chiều (tt)

Khi biến không tường minh, cấu trúc khai báo có dạng như sau:

```
typedef <kiểu cơ sở> <tên kiểu>[<N1>][<N2>];  
  
<tên kiểu> <tên biến>;  
<tên kiểu> <tên biến 1>, <tên biến 2>;
```

Ví dụ:

```
typedef int MaTran10x20[10][20];  
typedef int MaTran5x10[5][10];  
  
MaTran10x20 a, b;  
MaTran11x11 c;  
MaTran10x20 d;
```

Truy xuất phần tử mảng 2 chiều

- Cũng như mảng 1 chiều, không thể truy xuất trực tiếp biến mảng 2 chiều.
- Truy xuất phần tử mảng 2 chiều

định_danh[vị_trí_hàng][vị_trí_cột]

- vị_trí_hàng: là hằng nguyên hoặc biểu thức nguyên có giá trị nhỏ hơn số hàng.

– vị_trí_cột: là hằng nguyên hoặc biểu thức nguyên có giá trị nhỏ hơn số cột.

```
int a[2][3] = {{1, 2, 3}, {4, 5, 6}};  
int b = a[0][0]; // truy xuất phần tử đầu tiên của mảng a  
a[0][0] = 0; // gán giá trị cho phần tử đầu tiên của mảng a
```

CHUỖI KÝ TỰ

Chuỗi ký tự

❖ Khái niệm

- Kiểu char chỉ chứa được một ký tự. Để lưu trữ một chuỗi (nhiều ký tự) ta sử dụng mảng (một chiều) các ký tự.
- Chuỗi ký tự kết thúc bằng ký tự '\0' (null)
→ Độ dài chuỗi = kích thước mảng - 1

❖ Ví dụ

```
char hoten[30]; //chua 29 ky tu  
char mssv[15]; //chua 14 ky tu
```

Chuỗi ký tự (tt)

❖ Khai báo

- Cách 1:

```
char <tên chuỗi>[<số ký tự tối đa>];
```

Ví dụ:

```
| char hoten[30];//chứa 29 ký tự  
| char mssv[15]; //chứa 14 ký tự
```

- Cách 2:

```
char *<tên chuỗi>;  
<tên chuỗi> = new char;
```

Ví dụ:

```
| char *hoten;  
| hoten = new char;
```

Chuỗi ký tự (tt)

❖ Khởi tạo

- Độ dài cụ thể:

```
char s[12] = { 'L', 'a', 'p', ' ', 't', 'r', 'i', 'n', 'h', '\0' };
char s[12] = "Lap trinh"; //tu dong them '\0'
```



- Tự động xác định độ dài:

```
char s[] = { 'L', 'a', 'p', ' ', 't', 'r', 'i', 'n', 'h', '\0' };
char s[] = "Lap trinh"; //tu dong them '\0'
```



Chuỗi ký tự (tt)

❖ Nhập / xuất chuỗi

- Sử dụng cách nhập xuất thông thường: cin và cout

⇒ Chỉ nhận được các ký tự từ bàn phím cho đến khi nhận ký tự khoảng trắng hay ký tự xuống dòng. Chuỗi kết quả không bao gồm ký tự khoảng trắng hay xuống dòng.

Chuỗi ký tự (tt)

```
void main()
{
    char *s;
    s=new char;
    cout<<"Nhap chuoi: ";
    cin>>s;
    cout<<"Xuat chuoi: "<<s;
    cout<<endl;
}
```

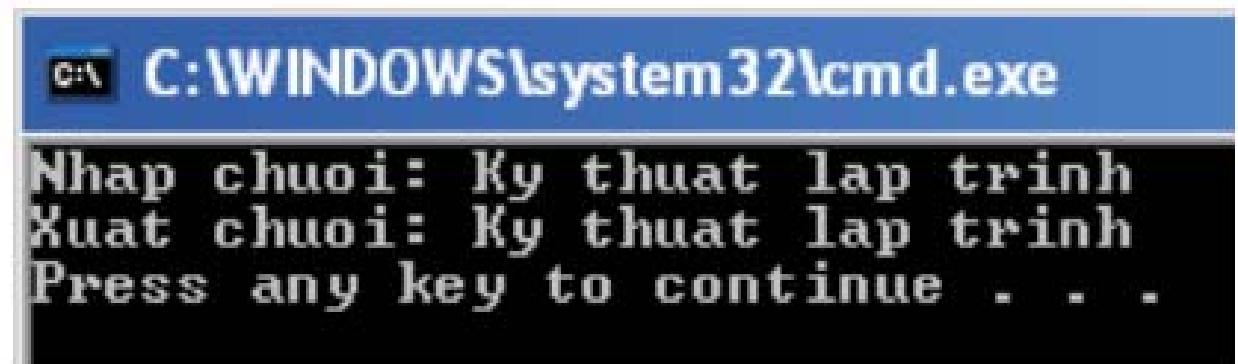
C:\WINDOWS\system32\cmd.exe

Nhap chuoi: Ky thuат lap trinh
Xuat chuoi: Ky
Press any key to continue . . .

Chuỗi ký tự (tt)

- Sử dụng hàm gets() và puts()
⇒ Nhận được trọn vẹn chuỗi nhập vào.

```
void main()
{
    char *s;
    s=new char;
    cout<<"Nhap chuoi: ";
    gets(s);
    cout<<"Xuat chuoi: "<<s;
    //puts(s);
    cout<<endl;
}
```



```
C:\WINDOWS\system32\cmd.exe
Nhap chuoi: Ky thuat lap trinh
Xuat chuoi: Ky thuat lap trinh
Press any key to continue . . .
```

Một số thao tác trên chuỗi ký tự

- ❖ Lấy chiều dài chuỗi

```
int *strlen(char *s)
```

⇒ Trả về chiều dài của chuỗi.

```
void main()
{
    char* s1;
    s1=new char [100];
    int l;
    cout<<"Nhap chuoi: ";
    gets(s1);
    l=strlen(s1);
    cout<<"Chieu dai: "<<l;
    cout<<endl;
}
```

C:\WINDOWS\system32\cmd.exe

Nhap chuoi: Kỹ thuật lập trình

Chieu dai: 18

Press any key to continue . . .

Một số thao tác (tt)

❖ Sao chép chuỗi

char *strcpy(char *dest, char *source)

⇒ Sao chép từ chuỗi source vào chuỗi dest.

```
void main()
{
    char *s1="Ky thuат lập trình", *s2;
    s2=new char [100];
    strcpy(s2,s1);
    cout<<"Chuoi vua copy: ";
    puts(s2);
    cout<<endl;
}
```

C:\WINDOWS\system32\cmd.exe

Chuoi vua copy: Ky thuật lập trình

Press any key to continue . . .

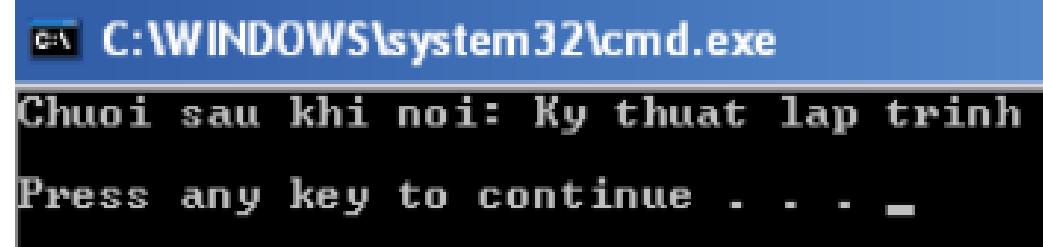
Một số thao tác (tt)

❖ Nối chuỗi

```
char *strcat(char *dest, char *source)
```

⇒ Nối chuỗi source vào chuỗi dest.

```
void main()
{
    char *s1="Ky thuат ", *s2="lap trinh", *s3;
    int l=strlen(s1)+strlen(s2);
    s3=new char [l];
    s3[0]='\0';
    strcat(s3,s1);
    strcat(s3,s2);
    s3[l]='\0';
    cout<<"Chuoi sau khi noi: ";
    puts(s3);
    cout<<endl;
}
```



The screenshot shows a Windows command prompt window titled 'C:\WINDOWS\system32\cmd.exe'. The text inside the window reads:

```
Chuoi sau khi noi: Ky thuat lap trinh
Press any key to continue . . .
```

BÀI TẬP THỰC HÀNH

1. Viết chương trình nhập vào ma trận M dòng N cột ($M < 100$, $N < 100$)

- a. In kết quả ma trận vừa nhập ra màn hình theo dòng cột
- b. Viết hàm in các phần tử dòng thứ i (i là tham số)
- c. Viết hàm tính tổng các phần tử dòng thứ i (i là tham số)
- d. Viết hàm in các phần tử cột thứ j (j là tham số)
- e. Viết hàm tính tổng các phần tử cột thứ j (j là tham số)
- f. Viết hàm in các phần tử chẵn trong ma trận
- g. Viết hàm in các phần tử lẻ trong ma trận
- h. Viết hàm in các phần tử là số nguyên tố trong ma trận

2. Viết chương trình nhập vào ma trận m dòng, n cột ($m < 100$, $n < 100$)

- a. Với mỗi dòng, sắp xếp các phần tử theo thứ tự tăng dần
- b. Với mỗi dòng, sắp xếp các phần tử theo thứ tự giảm dần
- c. Với mỗi cột, sắp xếp các phần tử theo thứ tự tăng dần
- d. Với mỗi cột, sắp xếp các phần tử theo thứ tự giảm dần

3. Viết chương trình nhập vào 1 ma trận m hàng và n cột

- a/. In các phần tử của ma trận
- b/ Tìm số lớn nhất và nhỏ nhất của ma trận
- c/. Đếm và in số phần tử chẵn trong ma trận
- d/. Đếm và in số phần tử lẻ trong ma trận
- e/. Đếm và in tất cả các phần tử là số nguyên tố trong ma trận
- f/. Sắp xếp các phần tử của mảng theo thứ tự tăng dần.

Bài3. Viết chương trình nhập vào 1 ma trận m hàng và n cột

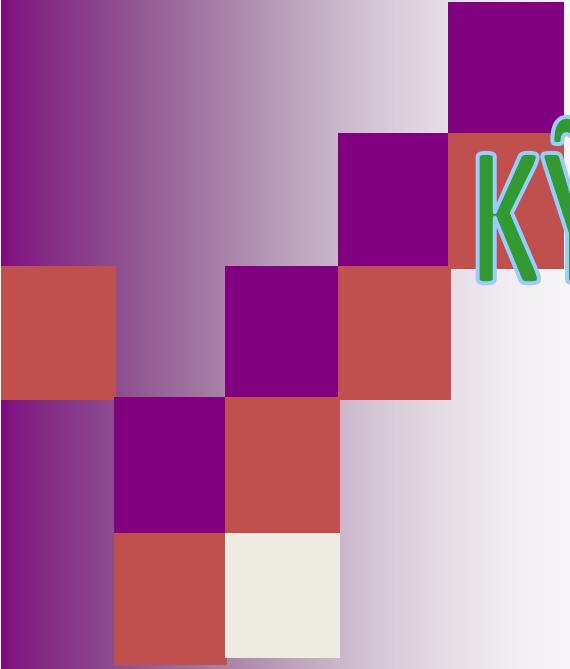
- a/. In các phần tử của ma trận
- b/ Tìm số lớn nhất và nhỏ nhất của ma trận
- c/. Đếm và in số phần tử chẵn trong ma trận
- d/. Đếm và in số phần tử lẻ trong ma trận
- e/. Đếm và in tất cả các phần tử là số nguyên tố trong ma trận
- f/. Sắp xếp các phần tử của mảng theo thứ tự tăng dần.

Bài 4: Hãy nhập vào hai ma trận A,B cùng số hàng và số cột.

- a/ Tính tổng, hiệu, tích của hai ma trận.
- b/ Gộp ma trận A, B thành ma trận C

Bài 5:Hãy nhập vào 16 số nguyên. In ra thành 4 dòng, 4 cột.

Trường ĐH Khoa Học Tự Nhiên
Khoa Vật Lý – Bộ Môn Vật Lý Ứng Dụng



KỸ THUẬT LẬP TRÌNH C++

Chương 5: CON TRỎ - CẤP PHÁT BỘ NHÓ ĐỘNG

Địa chỉ và toán tử lấy địa chỉ

❖ Biến

- Liên quan đến biến: tên biến, kiểu biến, giá trị của biến
- Địa chỉ của biến là số thứ tự của byte đầu tiên trong dãy các byte mà bộ nhớ cấp phát.
- Để xem địa chỉ của biến, sử dụng toán tử lấy địa chỉ &

❖ Ví dụ

- int a = 5;
⇒ biến có tên a, thuộc kiểu int và có giá trị = 5
⇒ địa chỉ của biến a: &a

Địa chỉ của ô nhớ

- Địa chỉ của ô nhớ là **số thứ tự** của byte đầu tiên trong một dãy các byte liên tiếp. Địa chỉ này được tính ở hệ đếm 16.
- Ví dụ :

int a[4];

→ Mỗi biến kiểu **int** chiếm **2 bytes**, như vậy bộ nhớ sẽ dành **8 ô nhớ** cho biến a.

→ Và cách đánh địa chỉ như sau: giả sử địa chỉ của biến a[0] là **ffee**

a[0]	a[1]	a[2]	a[3]
ffee	ffff0	ffff2	ffff4

CON TRÓ

Con trỏ

❖ Khái niệm

- Con trỏ là một biến dùng để chứa địa chỉ.
- Vì có nhiều kiểu biến chứa trong các loại địa chỉ khác nhau, nên có nhiều kiểu con trỏ khác nhau: kiểu int, kiểu float...
- Phải khai báo biến con trỏ trước khi sử dụng

❖ Cú pháp

<kiểu dữ liệu> *<tên con trỏ>; Hay <kiểu dữ liệu>* <tên con trỏ>

- Có thể khai báo con trỏ nhiều cấp,
- ví dụ: int * *ppi;
 float * * *pppf;

Con trỏ (tt)

Ví dụ

```
int x, y, *px, *py, *pz;  
float z;  
//x, y là 2 biến số nguyên  
//px, py, pz là 3 con trỏ kiểu số nguyên  
//z là biến số thực
```

- Khi đó, các câu lệnh sau sẽ có ý nghĩa:

```
px = &x; //gán địa chỉ của biến x cho con trỏ px  
py = &y; //gán địa chỉ của biến y cho con trỏ py  
pz = &z; //sai???
```

Qui tắc sử dụng con trỏ

❖ Sử dụng tên con trỏ

- Con trỏ cũng là một biến, nên khi tên của con trỏ xuất hiện ở đâu thì giá trị của nó cũng sẽ được sử dụng ở đó.
- Khi tên con trỏ ở bên trái của một toán tử gán, thì giá trị của biểu thức bên phải phải là một địa chỉ để gán cho con trỏ này.
- Ví dụ:

```
int a, b, *p, *q, *w;  
p = &a; //gán địa chỉ của biến a cho con trỏ p  
q = p; //gán giá trị của con trỏ p cho con trỏ q  
w = b; //sai???
```

Phép lấy địa chỉ của một biến

- Kí hiệu: &
- Cách sử dụng: <tên con trỏ> = &<VarName>
- **Ví dụ:** int *pi=&a; // pi giữ địa chỉ của các biến nguyên a

☞ Lưu ý:

Trong trường hợp trên, pi là một địa chỉ có giá trị địa chỉ là &a, và &pi cũng là một địa chỉ nhưng giá trị địa chỉ này không phải là &a mà là nơi ghi nhận giá trị &a, khi đó:

int **ppi=π // con trỏ 2 cấp

Phép toán lấy giá trị tại một địa chỉ mà một con trỏ đang trỏ tới

- Kí hiệu: *
- Cách sử dụng: *<PointerName>
- Ví dụ:

```
int a= 10; // biến a có giá trị 10
```

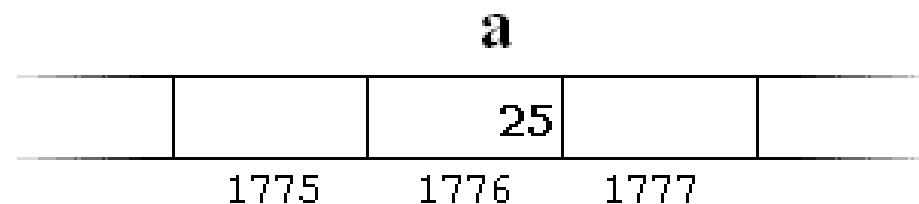
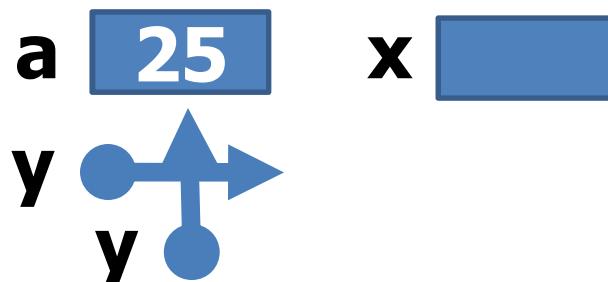
```
int *pi;
```

```
pi =&a; // pi giữ địa chỉ của biến a
```

→ *pi là giá trị của a bà bằng 10

Ví dụ 1

```
int a=25, x;  
int *y;  
x=a;  
y=&a;
```



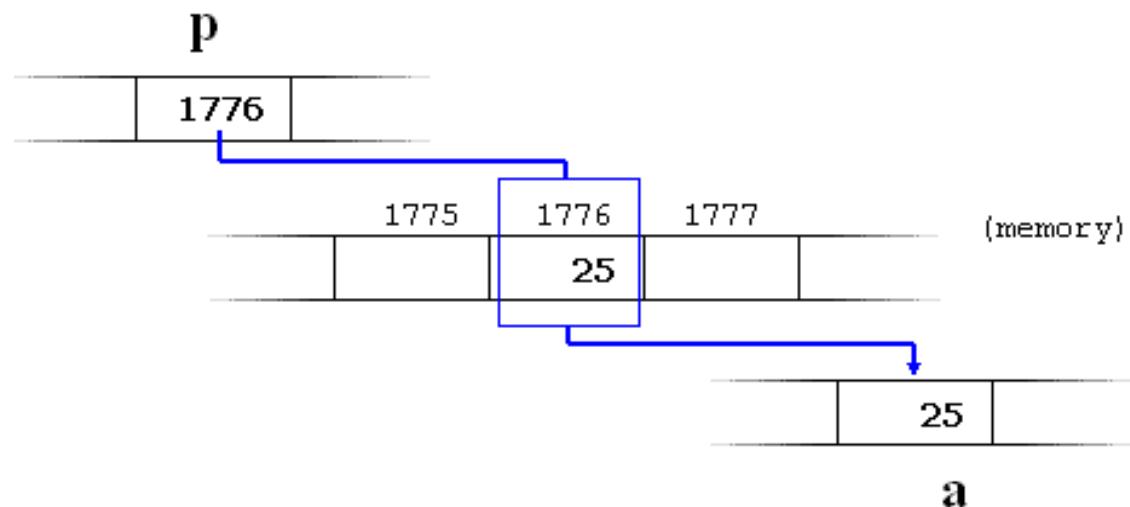
Ví dụ 2

```
int a, *p;
```

```
a=25;
```

```
p=&a;
```

```
m=*p ;
```



Cho biết Giá trị của m ?

Chú ý

- Cần chú ý rằng việc **thay đổi giá trị của pi** sẽ làm cho **giá trị của a thay đổi theo và ngược lại**.
- Khi ta khai báo: `int a=10;`

```
int *pi=&a; // pi giữ địa chỉ biến a  
*pi=*pi +2;
```

→ sẽ làm cho biến a có giá trị là 12

☞ **Lưu ý:** Việc sử dụng và thao tác trên giá trị tại địa chỉ của con trỏ chỉ được thực hiện sau khi con trỏ đã có địa chỉ

Các thao tác trên con trỏ

▪ Lệnh gán con trỏ

Có thể dùng phép gán để

-Gán giá trị của một con trỏ cho một con trỏ khác có cùng kiểu

- Gán con trỏ với địa chỉ một biến: `p = &x;`

-Gán con trỏ với con trỏ khác: `p = q;` (sau phép toán gán này p, q chứa cùng một địa chỉ, cùng trỏ đến một nơi).

Khi ta khai báo: `int x;`

`int *p1, *p2;`

`p1 = &x; // p1 giữ địa chỉ biến a`

`p2 = p1;`

→ cả hai p1 và p2 cùng trỏ đến biến x.

Các thao tác trên con trỏ (tt)

Khi gán con trỏ có một số lưu ý sau đây:

- + Tất cả các loại con trỏ đều có phép gán.
- + Phép gán với con trỏ yêu cầu về trái là 1 con trỏ và về phải là 1 địa chỉ
- + Phép gán yêu cầu sự tương xứng về kiểu dữ liệu, nếu ko tương xứng chúng ta phải ép kiểu.
- + Phép gán với 1 con trỏ kiểu void ko cần thiết phải tương xứng hoàn toàn về kiểu dữ liệu, void* có thể tương ứng với tất cả (như ở ví dụ chap trước), thậm chí là vượt cấp (vượt hẳn 2 cấp) như ví dụ sau

Các thao tác trên con trỏ (tt)

■ Phép toán số học trên con trỏ

- Chỉ có 2 phép toán sử dụng trên con trỏ là phép cộng và trừ
- Khi cộng (+) hoặc trừ (-) 1 con trỏ với 1 số nguyên N; kết quả trả về là 1 con trỏ. Con trỏ này trở đến vùng nhớ cách vùng nhớ của con trỏ hiện tại một số nguyên lần kích thước của kiểu dữ liệu của nó.

Chú ý:

Việc thực hiện các phép tính số học với con trỏ hơi khác so với các kiểu dữ liệu số nguyên khác. Trước hết, chỉ phép cộng và trừ là được phép dùng. Nhưng cả cộng và trừ đều cho kết quả phụ thuộc vào kích thước của kiểu dữ liệu mà biến con trỏ trỏ tới.

Ví dụ:

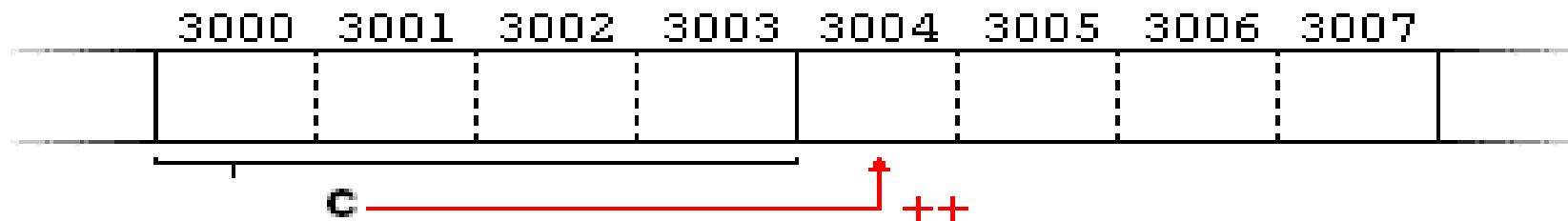
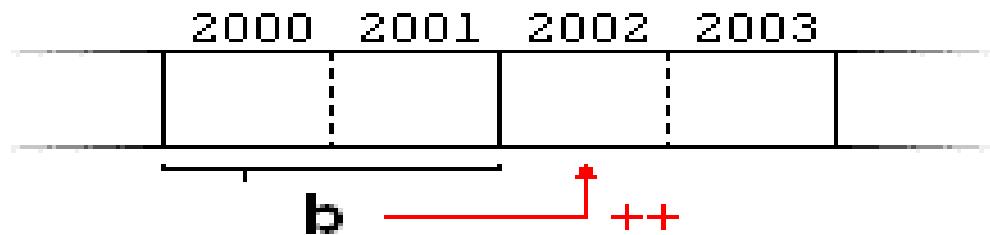
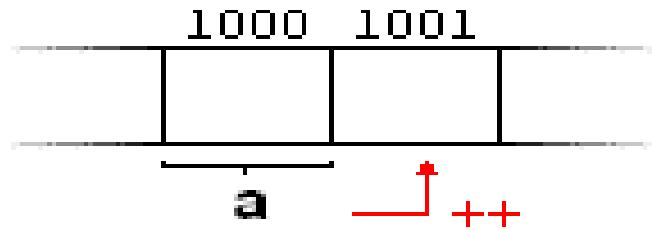
Các kiểu số nguyên, *char* chiếm 1 byte,
short chiếm 2 byte và *long* chiếm 4 byte.
Khi đó

```
char *a;  
short *b;  
long *c;
```

→ Do đó:

```
a = a + 1; // trỏ vào 1 byte tiếp theo  
b = b + 1; // trỏ vào 2 bytes tiếp theo  
c = c + 1; // trỏ vào 4 bytes tiếp theo
```

Minh họa



Ví dụ

```
#include <iostream.h>
#include<conio.h>
void main ()
{
    int a = 20, b = 15, *pa, *pb, temp;
    pa = &a; // con trỏ pa chứa địa chỉ của a
    pb = &b; // con trỏ pb chứa địa chỉ của b
    temp = *pa;
    *pa = *pb;
    *pb = temp;
    cout << "a = " << a << endl;
    cout << "b = " << b;
}
```

// kết quả xuất ra màn hình ?

a = 15
b = 20

Các thao tác trên con trỏ (tt)

- Các phép toán khác

- ❖ Phép toán tính khoảng cách giữa 2 con trỏ

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

- ❖ Phép so sánh: để kiểm tra 2 con trỏ có trỏ vào cùng 1 vùng nhớ hay không, hoặc kiểm tra 1 con trỏ có phải là đang trỏ vào NULL hay không

- ❖ Phép so sánh lớn hơn nhỏ hơn : > , < , >= , <= sử dụng để kiểm tra về độ thấp cao giữa 2 địa chỉ . Con trỏ nào nhỏ hơn thì trỏ vào địa chỉ thấp hơn.

- == != >>= <<=

★ Không thể thực hiện các phép toán: * / %

Con trỏ hằng, và hằng con trỏ

- **Hằng con trỏ:** những con trỏ mà chỉ trỏ cố định vào 1 vùng nhớ , những con trỏ này ko có khả năng trỏ vào vùng nhớ khác, ko thay đổi được
- **Con trỏ hằng:** những con trỏ mà trỏ vào 1 vùng nhớ cố định, con trỏ này chỉ có tác dụng trỏ đến, chứ không có khả năng thay đổi giá trị của vùng nhớ nà



Con trỏ và mảng 1 chiều

Mảng một chiều

- Trong C++, một mảng được coi là một kiểu con trỏ hàng, được định vị tại một vùng nhớ xác định và địa chỉ của tên mảng trùng với địa chỉ của phần tử đầu tiên của mảng.
- Ví dụ khai báo:

```
int A[5];
```

Địa chỉ của mảng A sẽ trùng với địa chỉ phần tử đầu tiên của mảng A , nghĩa là:

```
A = &A[0];
```

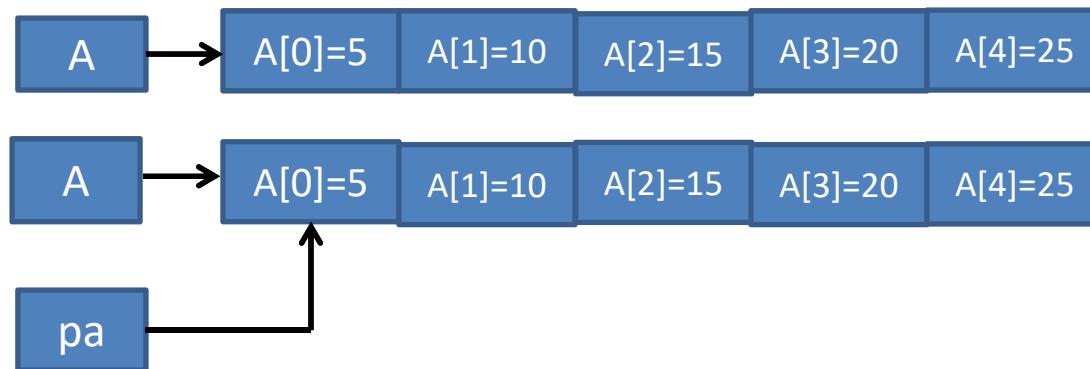
Quan hệ giữa con trỏ và mảng

Vì tên của mảng được coi như một con trỏ hằng, nên nó có thể được gán cho một con trỏ có cùng kiểu.

- **Ví dụ khai báo:**

```
int A[5] = {5, 10, 15, 20, 25};
```

```
int *pa = A;
```



// kết quả : pa = &A[0];

*pa = 5;

Phép toán trên con trỏ và mảng

Khi một con trỏ trỏ đến mảng, thì các phép toán tăng hay giảm trên con trỏ sẽ tương ứng với phép dịch chuyển trên mảng.

- **Ví dụ khai báo:**

`int A[5] = {5, 10, 15, 20, 25};`

`int *pa = &A[2]; // con trỏ pa sẽ trỏ đến địa chỉ của phần tử A[2] và giá trị của pa là: *pa = A[2] = 15.`

`pa = pa + 1; // Sẽ đưa con trỏ pa trỏ đến địa chỉ của phần tử tiếp theo trong mảng A, phần tử A[3].`

`pa = pa - 2; //Sẽ đưa con trỏ pa đến địa chỉ của phần tử A[1]`

Con trả hàm

- Khi 1 chương trình chạy thì các hàm nằm bên chương trình đó được load lên không gian nhớ ảo, chúng nằm trong vùng nhớ code.
- Bản chất của con trả hàm cũng là 1 con trả có định kiểu. Ta có thể sử dụng con trả hàm để gọi hàm khi đã biết địa chỉ của hàm
- Con trả hàm được dùng để truyền tham số có dạng hàm.

Khai báo con trỏ hàm

- Con trỏ hàm được khai báo tương tự như khai báo nguyên mẫu hàm thông thường trong C++, ngoại trừ việc có thêm kí hiệu con trỏ “*” trước tên hàm.
- Cú pháp khai báo con trỏ hàm như sau:

<Kiểu dữ liệu trả về> (*<Tên hàm>)(<Các tham số>);

Ví dụ:

*int (*Calcul)(int a, int b);*

Lưu ý

- Dấu “()” bao bọc tên hàm là cần thiết để chỉ ra rằng ta đang khai báo một con trả hàm.

Nếu không có dấu ngoặc đơn này, trình biên dịch sẽ hiểu rằng ta đang khai báo một hàm thông thường và có giá trị trả về là một con trả.

Ví dụ, hai khai báo sau là khác nhau hoàn toàn:

*int (*Calcul)(int a, int b); // Khai báo một con trả hàm*

*int *Calcul(int a, int b); // Khai báo một hàm trả về kiểu
con trả*

Sử dụng con trỏ hàm

Con trỏ hàm được dùng khi cần gọi một hàm như là tham số của một hàm khác. Khi đó, một hàm được gọi phải có khuôn mẫu giống với con trỏ hàm đã được khai báo.

- **Ví dụ:**

- ❖ *int (*Calcul)(int a, int b);*
→ gọi các hàm có hai tham số kiểu int và trả về cùng kiểu int như sau:
 - *int add(int a, int b);*
 - *int sub(int a, int b);*

CẤP PHÁT BỘ NHỚ ĐỘNG

CẤP PHÁT BỘ NHỚ ĐỘNG

Ý nghĩa của việc cấp phát động bộ nhớ

- Biến toàn cục (*global variables*) được cấp phát bộ nhớ vào lúc biên dịch. Biến cục bộ (*local variables*) dùng stack. Tuy nhiên, biến toàn cục hay cục bộ không thể được tạo thêm trong khi thực thi chương trình. Một số chương trình cần thêm bộ nhớ khi thực thi → **giải pháp cho vấn đề này là cấp phát động.**
- Cấp phát động là phương tiện nhờ đó một chương trình có thể **dành được thêm bộ nhớ** trong khi đang thực thi.
- Con trỏ cung cấp sự hỗ trợ cho cấp phát bộ nhớ động trong C/C++.

Cấp phát bộ nhớ động

C++ cung cấp hai toán tử cấp phát và thu hồi bộ nhớ động: **new** và **delete**.

- Toán tử **new** cấp phát bộ nhớ và trả về một con trỏ đến byte đầu tiên của vùng nhớ được cấp phát.
- Toán tử **delete** thu hồi vùng nhớ được cấp phát trước đó bởi toán tử new.

Cú pháp: <pointer_var_name>=**new** <Type>;

Hoặc:

<pointer_var_name>=**new** <Type>[N_blocks];

delete <pointer_var_name>;

delete p;

Lưu ý:

- Cần kiểm tra xem việc cấp phát bộ nhớ có thành công hay không. Nếu thành công, con trỏ trỏ đến **địa chỉ đầu khối** nhớ được cấp phát và lúc này chúng ta mới có thể thực hiện các thao tác trên con trỏ này.
- Khi ta khai báo:

```
int *pi;  
pi= new int;  
if (pi==NULL)  
{  
    cout<<"khong du bo nho";  
    exit(0); /* kết thúc CTr, trả về mã lỗi Code (0) cho  
hệ thống*/  
}
```

Ví dụ:

```
#include <iostream.h>
int main()
{
    int *p;
    p = new int;
    if (p==NULL)
    {
        cout<<"khong du bo nho";
        exit(0);
    }
    *p = 100;
    cout << "At " << p << " ";
    cout << "is the value " << *p << "\n";
    delete p;
    return 0;
}
```

Lưu ý khi cấp phát bộ nhớ động

- ❖ Một con trỏ, sau khi bị giải phóng địa chỉ, vẫn có thể được cấp phát một vùng nhớ mới hoặc trỏ đến một địa chỉ mới:

```
int *pa = new int(12); // Khai báo con trỏ pa, cấp  
                      // phát bộ nhớ và gán giá trị ban đầu cho pa là 12.  
delete pa; // Giải phóng vùng nhớ vừa cấp cho pa.  
int A[5] = {5, 10, 15, 20, 25};  
pa = A; // Cho pa trỏ đến địa chỉ của mảng A
```

Lưu ý (tt)

- Nếu có nhiều con trỏ cùng trỏ vào một địa chỉ, thì chỉ cần giải phóng bộ nhớ của một con trỏ, tất cả các con trỏ còn lại cũng bị giải phóng bộ nhớ:

```
int *pa = new int(12); // *pa = 12
int *pb = pa;    // pb trỏ đến cùng địa chỉ pa.
*pb += 5;      // *pa = *pb = 17
delete pa;    // Giải phóng cả pa lẫn pb
```

Lưu ý (tt)

Một con trỏ sau khi cấp phát bộ nhớ động bằng thao tác new, cần phải phóng bộ nhớ trước khi trả đến một địa chỉ mới hoặc cấp phát bộ nhớ mới:

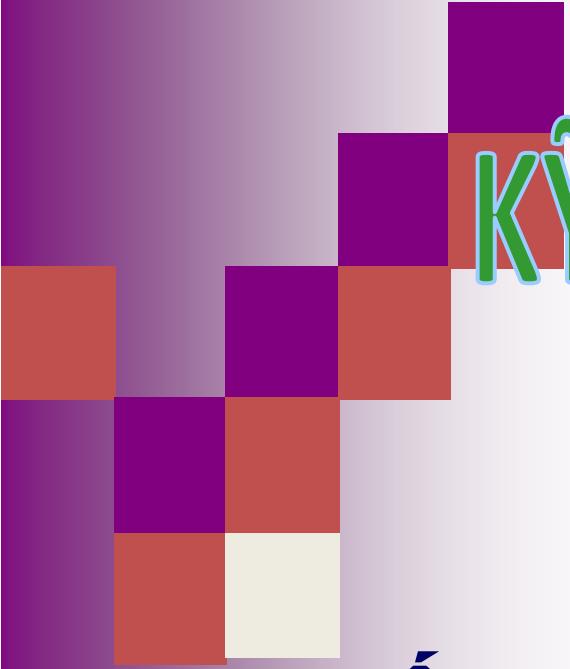
Ví dụ:

```
int *pa = new int(12); // pa được cấp bộ nhớ và *pa = 12  
*pa = new int(15); // pa trả đến địa chỉ khác và *pa = 15.  
// địa chỉ cũ của pa vẫn bị coi là bận
```

Bài tập con trỏ

1. Viết chương trình nhập vào mảng các số nguyên, sau đó sử dụng con trỏ để gán giá trị các thành phần của mảng về 0.
2. Tìm vị trí của phần tử nhỏ nhất trong mảng con trỏ pa (pa là tên mảng con trỏ đó).
3. Tìm vị trí của phần tử lớn nhất trong ma trận con trỏ pa.
4. Viết chương trình nhập số nguyên dương n gồm k chữ số ($0 < k \leq 5$) , sắp xếp các chữ số của n theo thứ tự tăng dần.
Ví dụ:
 - Nhập n = 1536
 - Kết quả sau khi sắp xếp: 1356.

Trường ĐH Khoa Học Tự Nhiên
Khoa Vật Lý – Bộ Môn Vật Lý Ứng Dụng



KỸ THUẬT LẬP TRÌNH C++

Chương 6: CẤU TRÚC DỮ LIỆU

MỞ ĐẦU

Để lưu giữ các giá trị gồm nhiều thành phần dữ liệu giống nhau → ta dùng kiểu mảng.

Thực tế có rất nhiều dữ liệu là tập các kiểu dữ liệu khác nhau, để quản lý kiểu dữ liệu này C++ đưa ra kiểu dữ liệu mới → **kiểu dữ liệu cấu trúc - struct.**

Ví dụ: bảng lý lịch cá nhân trong đó gồm nhiều kiểu dữ liệu như: họ tên, tuổi, giới tính,...

VÍ DỤ

❖ Thông tin 1 SV

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

❖ Yêu cầu

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

VÍ DỤ (tt)

❖ Khai báo các biến để lưu trữ 1 SV

- char mssv[13]; // “3.01.02.1234”
- char hoten[30]; // “Nguyen Van A”
- char ntns[8]; // “29/12/82”
- char phai; // ‘y’ ⇔ Nam, ‘n’ ⇔ Nữ
- float toan, ly, hoa; // 8.5 9.0 10.0

❖ Truyền thông tin 1 SV cho hàm

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

VÍ DỤ (tt)

❖ Nhận xét

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

❖ Ý tưởng

- Gom những thông tin của cùng 1 SV thành một kiểu dữ liệu mới => Kiểu dữ liệu cấu trúc (struct)

Khai báo kiểu cấu trúc và biến cấu trúc

❖ Cú pháp 1

//khai bao kieu cau truc va bien cau truc

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

//khai bao bien cau truc rieng

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



Các biến được khai báo cũng có thể đi kèm khởi tạo:

<tên cấu trúc> <tên biến> = { giá trị khởi tạo } ;

Ví dụ khai báo

- ❖ Ví dụ 1: xây dựng cấu trúc Điểm trong không gian 2 chiều

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

Hoặc

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

struct DiemOxy diem1, diem2;
```

Khai báo (tt)

❖ Cú pháp 2

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

<tên kiểu cấu trúc> <tên biến>;
```

Ví dụ

- ❖ Ví dụ 2: xây dựng cấu trúc Sinh viên để lưu trữ thông tin sinh viên

```
typedef struct
{
    char MSSV[13];
    char HoTen[30];
    char phai;
    char NTNS[8];
    float Toan, Van, TrungBinh;
} SinhVien;

SinhVien sv;
```

Truy xuất dữ liệu cấu trúc

❖ Đặc điểm

- Không thể truy xuất trực tiếp vào các thành phần trong cấu trúc, mà phải thông qua toán tử thành phần `.` hay còn gọi là toán tử chấm (dot operation)

❖ Cú pháp

 <tên biến cấu trúc>.<tên thành phần>

❖ Đối với biến con trả:

<tên biến cấu trúc> → <tên thành phần>

- Ví dụ 3: Nhập vào tọa độ 2 điểm trong không gian 2 chiều. Tính khoảng cách giữa 2 điểm này

Ví dụ

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

void Nhap(DIEM &d)
{
    cout<<"Nhập hoành độ: ";
    cin>>d.x;
    cout<<"Nhập tung độ: ";
    cin>>d.y;
}

void Xuat(DIEM d)
{
    cout<< "("<<d.x<< ", "<<d.y<< ")"<<endl;
}
```

Phép toán gán cấu trúc

Cũng giống các biến mảng, để làm việc với một biến cấu trúc chúng ta phải thực hiện thao tác trên từng thành phần của chúng. Ví dụ vào/ra một biến cấu trúc phải viết câu lệnh vào/ra từng cho từng thành phần.

Cấu trúc:

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

VÍ DỤ

struct DIEM

```
{  
    int      x, y;  
  
} diem1 = {2912, 1706}, diem2;  
...  
diem2 = diem1;  
  
diem2.x = diem1.x;  
  
diem2.y = diem1.y * 2;
```

Chú ý: không gán bộ giá trị cụ thể cho biến cấu trúc.
Cách gán này chỉ thực hiện được khi khởi tạo.

VÍ DỤ (tt)

Ví dụ 2: Nhập vào hai phân số và tính tổng của hai phân số

```
#include <iostream.h>
#include <conio.h>
struct Phanso {
    int tu ;
    int mau ;
} a, b, c ;

void main()
{
    clrscr();
    cout << "Nhập phân số a:" << endl ;           // nhập a
    cout << "Tử:"; cin >> a.tu;
    cout << "Mẫu:"; cin >> a.mau;
    cout << "Nhập phân số b:" << endl ;           // nhập b
    cout << "Tử:"; cin >> b.tu;
    cout << "Mẫu:"; cin >> b.mau;
    cout << "Tổng là:" << endl ;
    cout << "Tử:"; cout << a.tu * b.mau + b.tu * a.mau;
    cout << "Mẫu:"; cout << a.mau * b.mau;
}
```

VÍ DỤ (tt)

```
cout << "Nhập phân số b:" << endl ;           // nhập b
cout << "Tử:"; cin >> b.tu;
cout << "Mẫu:"; cin >> b.mau;
c.tu = a.tu*b.mau + a.mau*b.tu;                // tính và in a+b
c.mau = a.mau*b.mau;
cout << "a + b = " << c.tu << "/" << c.mau;
getch();
}
```

Kích thước của cấu trúc

Để trả lại kích thước cho kiểu cấu trúc, ta dùng hàm **sizeof**

Cú pháp:

Sizeof (<tên cấu trúc>)

Ví dụ:

```
struct A
{
    int x;
    double y;
};
sizeof(A) = ???
```

Các lưu ý về cấu trúc

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

```
struct DIEM { float x, y; } d1;  
float temp; scanf("%f", &temp); d1.x = temp;
```

Kiểu hợp nhất - Union

- **Khái niệm**

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

- **Khai báo**

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

Truy nhập thành phần của kiểu hợp nhất

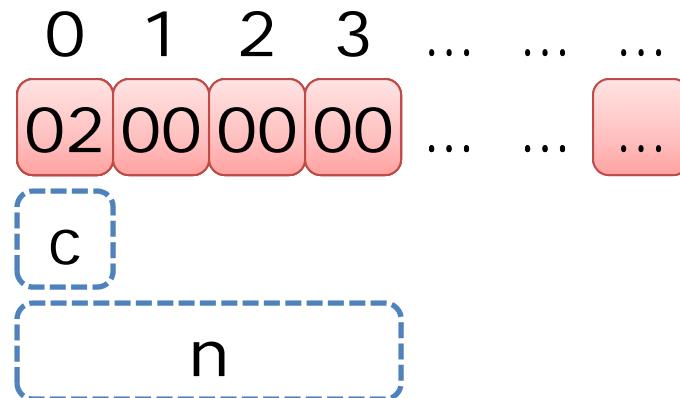
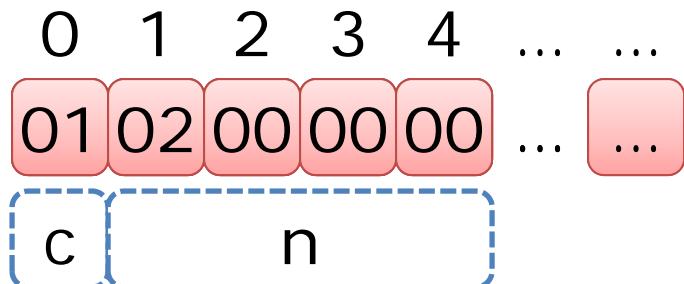
Cú pháp truy cập đến các thành phần của hợp cũng tương tự như kiểu cấu trúc, tức cũng sử dụng toán tử lấy thành phần (đầu chấm . hoặc → cho biến con trả kiểu hợp).

So sánh struct và union

- Ví dụ

```
struct MYSTRUCT  
{  
    char c;  
    int n;  
} s;  
  
s.c = 1; s.n = 2;
```

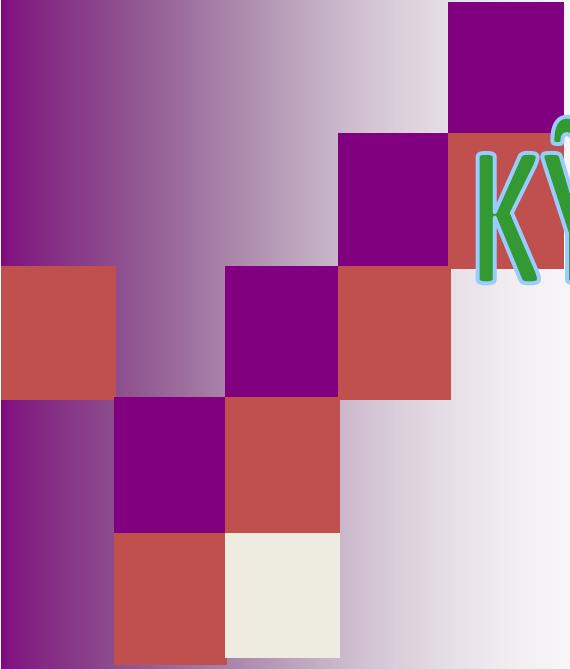
```
union MYUNION  
{  
    char c;  
    int n;  
} u;  
  
u.c = 1; u.n = 2;
```



Thực hành

1. Viết chương trình nhập và xuất ra hai phân số. Từ đó tính hiệu, tích, thương của hai phân số đó.
2. Qui đồng mẫu số hai phân số.
3. Nhập và xuất ba cạnh của tam giác. Tính chu vi, diện tích của tam giác.
4. Nhập mảng gồm 5 phân số. Sắp xếp các số theo thứ tự tăng dần.

Trường ĐH Khoa Học Tự Nhiên
Khoa Vật Lý – Bộ Môn Vật Lý Ứng Dụng



KỸ THUẬT LẬP TRÌNH C++

Chương 7:

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG LỚP

1. Khái niệm Lập Trình Hướng Đối Tượng

- Lập trình cấu trúc dẫn đến khái niệm trừu tượng hóa:
 - Không quan tâm đến các chi tiết không quan trọng bên trong.
 - Không quan tâm việc thực hiện của chương trình.
 - Chỉ quan tâm đến kết quả.
→ **Lập trình hướng đối tượng**
- Khái niệm:
 - Lập trình hướng đối tượng được xây dựng trên nền tảng của khái niệm lập trình có cấu trúc và sự trừu tượng hóa dữ liệu.

- Là phương pháp lập trình:
 - Mô tả chính xác các đối tượng trong thế giới.
 - Lấy đối tượng làm nền tảng xây dựng thuật toán.
 - Thiết kế xoay quanh dữ liệu của hệ thống.
 - Chương trình được chia thành các lớp đối tượng.
 - Dữ liệu được đóng gói, che dấu và bảo vệ.
 - Đối tượng làm việc với nhau qua thông báo.
 - Chương trình được thiết kế theo cách từ dưới lên (bottom-up) và từ trên xuống (top – down).

• **Dữ liệu và giao**

Đối tượng = dữ liệu + phương thức

Là khái niệm trừu tượng phản ánh các thực thể trong thế giới thực

- ❖ Có thể là một thực thể vật lý.
- ❖ Có thể là một khái niệm trừu tượng.
- Một đối tượng là sự đóng gói 2 thành phần:
 - ❖ Trạng thái (state) dữ liệu hay thuộc tính (attribute): đó là các đặc điểm của đối tượng.
 - ❖ Các ứng xử (behavior) hay hành vi, thao tác, phương thức: đó là một hành động mà đối tượng có thể thực hiện.
- Có hai loại đối tượng: đối tượng trừu tượng và đối tượng thực.
- Được định nghĩa là sự thể hiện của một lớp (sẽ biết ở phần sau). Là các thực thể trong hệ thống hướng đối tượng (sẽ biết ở phần sau).

- Ví dụ 1:

- Một con người là một đối tượng
 - Họ, tên, chiều cao, cân nặng... là thuộc tính của tất cả con người.
 - Hành vi của con người là ngồi, đứng, đi, chạy, nhảy...

- Ví dụ 2:

- Một sinh viên trừu tượng gồm các thuộc tính:
 - MSSV, HỌ TÊN, NGÀY SINH, QUÊ QUÁN...
- Một sinh viên thực:
 - 070042T, Nguyễn Văn A, 01/6/1983, TPHCM...

3. Lớp (class):

- a. **Định nghĩa:** Lớp là tập hợp các biến và các hàm thao tác các biến đó. Các hàm và các biến được khai báo được gọi là thành phần của lớp.
- b. **Cú pháp khai báo lớp:**

```
class tên_lớp
{
    // Khai báo các thành phần dữ liệu (thuộc tính)
    // Khai báo các phương thức (hàm)
};
```

(Thuộc tính của lớp có thể là biến, mảng, con trỏ,...)

Khi khai báo các thành phần, phương thức của lớp dùng các từ khóa **private**, **protected** và **public**. Việc dùng này nhằm qui định phạm vi sử dụng của các thành phần. Trong đó:

- **private** members: chỉ được truy nhập bởi các thành phần khác của lớp hoặc bạn của lớp.

-**protected** members: được truy nhập bởi các thành phần khác của lớp, của lớp kế thừa hoặc bạn của lớp.

-**public** members: được truy nhập ở mọi nơi mà đối tượng của lớp xuất hiện.

Khi đó cấu trúc khai báo lớp có dạng :

```
class <tên_lớp>
{
    private:
        <khai báo các thành phần riêng>;
    protected:
        <Khai báo các thành phần được bảo vệ>;
    public:
        <Khai báo các thành phần chung> ;
};
```

b. Khai báo thành phần của lớp

Thành phần của lớp

- Có thể gồm:

Dữ liệu

Thuộc tính

Phương thức

b. Khai báo (tt)

- Dữ liệu:

Tương tự như khai báo biến

```
<kiểu dữ liệu> <tên_thành_phần>;
```

Chú ý: không được khởi tạo giá trị ban đầu

b. Khai báo (tt)

■ Hàm thành phần

Cách 1: Khai báo trong lớp và định nghĩa ngoài lớp

```
<kiểu trả về> tênlớp::<tên_hàm>([đối số])
```

```
{
```

```
// <thân hàm>
```

```
}
```

Cách 2: định nghĩa ngay trong lớp

b. Khai báo thành phần của lớp (tt)

Ví dụ 1:

Xây dựng cấu trúc dữ liệu mô tả sinh viên:

Dữ liệu: họ tên, ngày sinh, giới tính, Điểm toán, lý, hóa, Đtb

Phương thức: nhập, tính đtb, in

→ Lớp sinh viên

b. Khai báo (tt)

Ví dụ 2:

Xây dựng cấu trúc dữ liệu mô tả các hóa đơn:

Dữ liệu: mã vật tư, tên vật tư, loại phiếu, ngày lập, khối lượng, đơn giá, thành tiền...

Phương thức: nhập, tính thành tiền, in

→ Lớp các hóa đơn

Ví dụ 3:

Xây dựng cấu trúc dữ liệu mô tả các phân số:

Dữ liệu: tử số, mẫu số

Phương thức: nhập, tối giản, in

→ Lớp các phân số

c. Khai báo đối tượng

Cú pháp:

```
<tên_lớp> <tên_đối_tượng>;
```

Ví dụ: khai báo 2 đối tượng sinh viên

```
SV sv1, sv2;
```

Khi đó sv1, sv2 là hai đối tượng sinh viên

d. Truy xuất thành phần

Dữ liệu

```
<đối_tượng>.<tp_dữ_liệu>;
```

Ví dụ: truy xuất họ tên và ngày sinh của sv

sv1.ht;

sv2.ns;

Nếu là con trỏ: <con_trỏ> → <tp_dữ_liệu>;

4. Các phương thức

Một đối tượng thường có 4 kiểu hành vi cơ bản

- ❖ Các phương thức khởi tạo: Constructor
- ❖ Các phương thức truy vấn: Queries
- ❖ Các phương thức cập nhập: Updates
- ❖ Các phương thức hủy: Destructor

a. Hàm khởi tạo

Khai báo:

```
<Tên_lớp>([ds tham số]);
```

Định nghĩa ngoài lớp:

```
<Tên_lớp>::<Tên_lớp>([ds tham số])  
{  
    //thân hàm  
}
```

a. Hàm khởi tạo (tiếp)

Như vậy hàm khởi tạo:

- ✓ Có với mọi lớp
- ✓ Tên hàm giống tên lớp
- ✓ Không có kiểu nêu không cần khai báo
- ✓ Không có giá trị trả về
- ✓ Nếu không xây dựng thì chương trình tự động sinh hàm khởi tạo mặc định
- ✓ Được gọi tự động khi khai báo thể hiện của lớp

a. Hàm khởi tạo (tiếp)

Một số hàm khởi tạo:

- Hàm khởi tạo mặc định (default constructor)
- Hàm khởi tạo sao chép (copy constructor)

Khai báo:

```
<đối_tượng>(const <đối_tượng> &<tham_số>)
```

Đối tượng mới sẽ là bản sao của đối tượng đã có

b. Hàm hủy - Destructor

Khai báo:

```
~<tên_lớp>();
```

Chức năng:

- Hủy bỏ, giải phóng các đối tượng khi nó hết phạm vi tồn tại

b. Hàm hủy (tiếp)

Như vậy hàm hủy:

- ✓ Không có đối số
- ✓ Không có giá trị trả về
- ✓ Không định nghĩa lại
- ✓ Trùng tên với lớp và có dấu ~ ở trước
- ✓ Thực hiện một số công việc trước khi hệ thống giải phóng bộ nhớ
- ✓ Chương trình dịch tự động sinh hàm hủy mặc định

5. Nạp chồng toán tử

Cú pháp:

```
<kiểu trả về>operator<tên toán tử>([ds tham số])
```

Định nghĩa ngoài lớp:

```
<kiểu trả về><tên_lớp>::operator<tên toán tử>([ds tham số])  
{  
    //thân hàm  
}
```

3. Nạp chồng toán tử (tiếp)

Ví dụ:

- ✓ Nạp chồng toán tử +, * của lớp phân số
- ✓ Nạp chồng toán tử + vector

Danh sách các toán tử có thể nạp chồng:

+ - * / = < > += -= *= /= << >>
<<= >>= == != <= >= ++ -- % & ^ ! |
~ &= ^= |= && || %= [] () , ->* -> new
delete new[] delete[]

3. Nạp chồng toán tử (tiếp)

Chú ý:

- ✓ Chỉ có thể định nghĩa lại các toán tử ở trên
- ✓ Không làm thay đổi độ ưu tiên của các toán tử
- ✓ Với toán tử 2 ngôi: toán tử bên trái là ẩn
toán tử bên phải là đối số

Do đó: số tham số bằng số toán hạng - 1

3. Nạp chồng toán tử (tiếp)

Cách gọi hàm toán tử:

✓ Dùng như cú pháp thông thường của phép toán

Ví dụ: PS a,b,c; c=a+b;

✓ Dùng như hàm thành phần của đối tượng

Ví dụ:

PS a,b,c;

c=a.operator+(b);