



---

# **Nhập môn lập trình**

## **Bài 1- Các khái niệm cơ bản**

---

# Mục tiêu

---



- Hiểu được tổng quan ngôn ngữ lập trình C/C++
- Công cụ lập trình
- Cấu trúc và cách thực thi chương trình
- Tập ký tự, từ khóa, quy tắc đặt tên
- Câu lệnh, chú thích
- Kiểu dữ liệu cơ sở
- Biến, hằng, biểu thức
- Toán tử, ép kiểu
- Các hàm thư viện C/C++ chuẩn



# 1. Lịch sử của ngôn ngữ C/C++

---

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

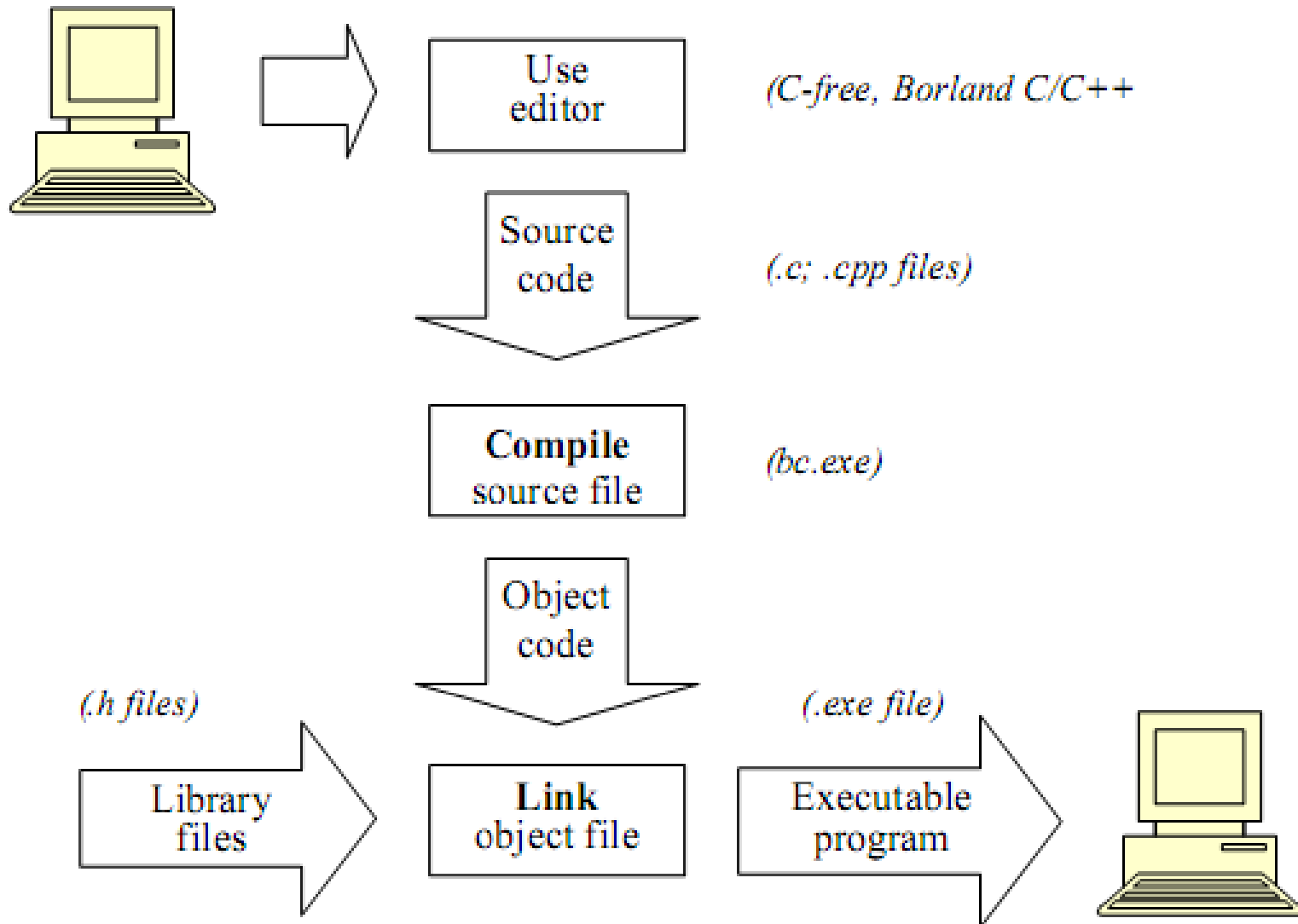


## 2. Kỹ thuật để giải quyết một bài toán

---

- ❑ Một chương trình máy tính được thiết kế để giải quyết một bài toán nào đó. Vì vậy, những bước cần để tìm kiếm lời giải cho một bài toán cũng giống như những bước cần để viết một chương trình.
- ❑ Các bước gồm:
  - Xác định yêu cầu của bài toán
  - Đưa ra thuật toán (dùng mã giả, hoặc lưu đồ)
  - Cài đặt (viết) chương trình
  - Thực hiện chương trình và kiểm chứng

# 3. Các bước trong chu trình phát triển chương trình



# 3.Các bước trong chu trình phát triển chương trình

---



- ❑ Nhập mã nguồn (source code)
  - Mã nguồn là tập lệnh dùng để chỉ dẫn máy tính thực hiện công việc do người lập trình đưa ra
  - Tập tin mã nguồn có phần mở rộng .cpp (C++)
- ❑ Biên dịch mã nguồn (compile)
  - Chương trình viết bằng ngôn ngữ cấp cao C/C++ được biên dịch sang mã máy bằng một chương trình dịch(compiler)

# 3. Các bước trong chu trình phát triển chương trình

---



- ❑ Liên kết các tập tin đối tượng tạo các tập tin thực thi (executable file).
  - C/C++ có một thư viện hàm được tạo sẵn
  - Tập tin đối tượng do trình biên dịch tạo ra kết hợp với mã đối tượng để tạo tập tin thực thi, quá trình này được tạo bởi bộ liên kết (Linker)
- ❑ Thực hiện chương trình

### 3. Các bước trong chu trình phát triển chương trình

---



- ❑ Thực hiện chương trình
  - Chương trình nguồn được biên dịch và liên kết sẽ tạo nên tập tin thực thi và thực thi tại dấu nhắc hệ thống
  - Nếu chương trình có lỗi phải được chỉnh sửa và biên dịch lại.
  - Quá trình 4 bước sẽ được lập lại cho đến khi tập tin thực thi thực hiện đúng yêu cầu bài toán



## 4. Khảo sát một chương trình C/C++ đơn giản

---



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

## 4. Khảo sát một chương trình C/C++ đơn giản

---



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

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

*#include <iostream.h>:*

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

## 4. Khảo sát một chương trình C/C++ đơn giản

---



*int main():*

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

## 4. Khảo sát một chương trình C/C++ đơn giản

---



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

Đây là một lệnh nằm trong phần thân của hàm main

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



## 5. Các chú thích

---

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



## 5. Các chú thích

---

```
/* My second program in C/C++ with more comments
   Author: Novice programmer
   Date: 01/01/2008
*/
#include <conio.h>
#include <iostream.h>
int main()
{
    cout << "Hello World! "; // output Hello World!
    cout << "I hate C/C++."; // output I hate C/C++.
    getch();
    return 0;
}
```



## 6. Cấu trúc của một chương trình C/C++

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

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

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

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

*Khai báo các tập tin thư viện*

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

*Khai báo prototype của các hàm tự tạo*



## 6. Cấu trúc của một chương trình C/C++

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

*Hàm chính của chương trình*

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

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

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





## 7. Các tập tin thư viện thông dụng

---

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



## 7. Các tập tin thư viện thông dụng

---

□ Các tập tin thư viện thông dụng gồm:

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



## 7. Các tập tin thư viện thông dụng

---

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



---

# BIỂU THỨC (Expressions)



# 1. Khái niệm về biểu thức

---

- ☐ Biểu thức là một sự kết hợp giữa các toán tử (operator) và các toán hạng (operand) theo đúng một trật tự nhất định.
- ☐ Mỗi toán hạng có thể là một hằng, một biến hoặc một biểu thức khác.
- ☐ Trong trường hợp, biểu thức có nhiều toán tử, ta dùng cặp dấu ngoặc đơn ( ) để chỉ định toán tử nào được thực hiện trước.



## 2. Kiểu dữ liệu(Data type)

---

- ❑ C/C++ có các kiểu dữ liệu cơ sở:
  - Ký tự (char)
  - Số nguyên (int)
  - Số thực (float)
  - Số thực có độ chính xác gấp đôi (double)
  - Kiểu bool
  - Kiểu vô định (void).
- ❑ Kích thước và phạm vi của những kiểu dữ liệu này có thể thay đổi tùy theo loại CPU và trình biên dịch.



## 2. Kiểu dữ liệu(Data type)

---

- ❑ Kiểu char chứa giá trị của bộ mã ASCII (*Americican Standard Code for Information Interchange*). Kích thước là 1 byte.
- ❑ Kích thước của kiểu int là 16 bits (2 bytes) trên môi trường 16-bit như DOS và 32 bits (4 bytes) trên môi trường 32-bit như Windows 95
- ❑ Kiểu void dùng để khai báo hàm không trả về giá trị hoặc tạo nên các con trỏ tổng quát (generic pointers).



## 2. Kiểu dữ liệu(Data type)

Kiểu dữ liệu	Kích thước bằng bits	Phạm vi tối thiểu
char	8	-127 to 127
unsigned char	8	0 to 255
signed char	8	-127 to 127
int	16 or 32	-32,767 to 32,767
unsigned int	16 or 32	0 to 65,535
signed int	16 or 32	giống như int
short int hoặc short	16	-32,767 to 32,767
unsigned short int	16	0 to 65,535
signed short int	16	giống như short int
long int hoặc long	32	-2,147,483,647 to 2,147,483,647
signed long int	32	giống như long int
unsigned long int	32	0 to 4,294,967,295
float	32	Độ chính xác là 6 ký số
double	64	Độ chính xác là 10 ký số
long double	80	Độ chính xác là 10 ký số





### 3. Định danh (Identifier Name)

---

- ❑ Trong C/C++, tên biến, hằng, hàm,... được gọi là định danh
- ❑ Những định danh này có thể là 1 hoặc nhiều ký tự. Ký tự đầu tiên phải là một chữ cái hoặc dấu \_ (underscore), những ký tự theo sau phải là chữ cái, chữ số, hoặc dấu \_
- ❑ C/C++ phân biệt ký tự HOA và thường.
- ❑ Định danh không được trùng với từ khóa (keywords).



## 4. Từ khóa (keywords)

- ❑ Là những từ được dành riêng bởi ngôn ngữ lập trình cho những mục đích riêng của nó
- ❑ Tất cả các từ khóa trong C/C++ đều là chữ thường (lowercase).
- ❑ Danh sách các từ khóa trong C/C++

auto	break	case	char	const	continue	default	do
double	else	enum	extern	float	for	goto	if
int	long	register	return	short	signed	sizeof	static
struct	switch	typedef	union	unsigned	void	volatile	while

## 5. Biến (variables)

---



- ❑ Biến là định danh của một vùng trong bộ nhớ dùng để giữ một giá trị mà có thể bị thay đổi bởi chương trình.
- ❑ Tất cả biến phải được khai báo trước khi sử dụng.
- ❑ Cách khai báo:

**type variableNames;**

- *type*: là một trong các kiểu dữ liệu hợp lệ.
- *variableNames*: tên của một hay nhiều biến phân cách nhau bởi dấu phẩy.



## 5. Biến (variables)

---

- ❑ Ngoài ra, ta có thể vừa khai báo vừa khởi tạo giá trị ban đầu cho biến:

`type varName1=value, ... ,varName_n=value;`

- ❑ Ví dụ:

`float mark1, mark2, mark3, average = 0;`



## 6. Phạm vi của biến

---

### ❑ Biến cục bộ (local variables)

- Những biến được khai báo bên trong một hàm gọi là biến cục bộ.
- Các biến cục bộ chỉ được tham chiếu đến bởi những lệnh trong khối (block) có khai báo biến.
- Một khối được đặt trong cặp dấu { }.
- Biến cục bộ chỉ tồn tại trong khi khối chứa nó đang thực thi và bị hủy khi khối chứa nó thực thi xong.



## 6. Phạm vi của biến

---

Ví dụ:

```
void func1(void)  
{  
    int x;  
    x = 10;  
}
```

```
void func2(void)  
{  
    int x;  
    x = -199;  
}
```

## 6. Phạm vi của biến

---



### □ Tham số hình thức(formal parameters)

- Nếu một hàm có nhận các đối số truyền vào hàm thì nó phải khai báo các biến để nhận giá trị của các đối số khi hàm được gọi.
- Những biến này gọi là các tham số hình thức. Những biến này được sử dụng giống như các biến cục bộ.



## 6. Phạm vi của biến

---

Ví dụ:

```
int sum(int from, int to)
{
    int total=0;
    for(int i=from ; i<=to ; i++)
        total +=i;
    return total;
}
```





## 6. Phạm vi của biến

---

### ❑ Biến toàn cục (global variables)

- Biến toàn cục có phạm vi là toàn bộ chương trình.
- Tất cả các lệnh có trong chương trình đều có thể tham chiếu đến biến toàn cục.
- Biến toàn cục được khai báo bên ngoài tất cả hàm.



## 6. Phạm vi của biến

---

```
#include <iostream.h>

int gVar = 100;

void increase()
{ gVar = gVar + 1;}

void decrease()
{ gVar = gVar -1;}

void main()
{
    cout << "Value of gVar= " << gVar;  increase();
    cout << "After increased, gVar= " << gVar;  decrease();
    cout << "After decreased, gVar= " << gVar;
}
```



## 7. Từ khóa const

---

- ❑ Giá trị của biến thay đổi trong suốt quá trình thực thi chương trình.
- ❑ Để giá trị của biến không bị thay đổi, ta đặt trước khai báo biến từ khóa const.
- ❑ Thông thường ta dùng chữ HOA để đặt tên cho những biến này.

Ví dụ:

```
const int MAX = 200;
```



## 8. Hằng (constants)

---

- ❑ Hằng là những giá trị cố định (fixed values) mà chương trình không thể thay đổi. Mỗi kiểu dữ liệu đều có hằng tương ứng. Hằng còn được gọi là literals.
- ❑ Hằng ký tự được đặt trong cặp nháy đơn.  
Ví dụ: 'a'
- ❑ Hằng nguyên là những số mà không có phần thập phân.  
Ví dụ 100 , -100



## 8. Hằng (constants)

1. Hằng số thực yêu cầu một dấu chấm phân cách phần nguyên và phần thập phân.

Ví dụ: 123.45

2. Cách viết một số loại hằng số

Kiểu dữ liệu	Các ví dụ về hằng	Ghi chú
int	1, 123, 21000, 234	
long int	35000L, 34l	Có ký tự l hoặc L ở cuối
unsigned int	10000U, 987u, 40000U	Có ký tự u hoặc U ở cuối
float	123.23f, 4.34e-3F	Có ký tự f hoặc F ở cuối
double	123.23, 1.0, 0.9876324	
long double	1001.2L	Có ký tự l hoặc L ở cuối



## 8. Hằng chuỗi ký tự (string constants)

---

□ Hằng chuỗi ký tự là một tập các ký tự đặt trong cặp nháy kép "".

Ví dụ:

- "This is a string" //là một chuỗi.
- 'a' //là một hằng ký tự.
- "a" //là một hằng chuỗi.



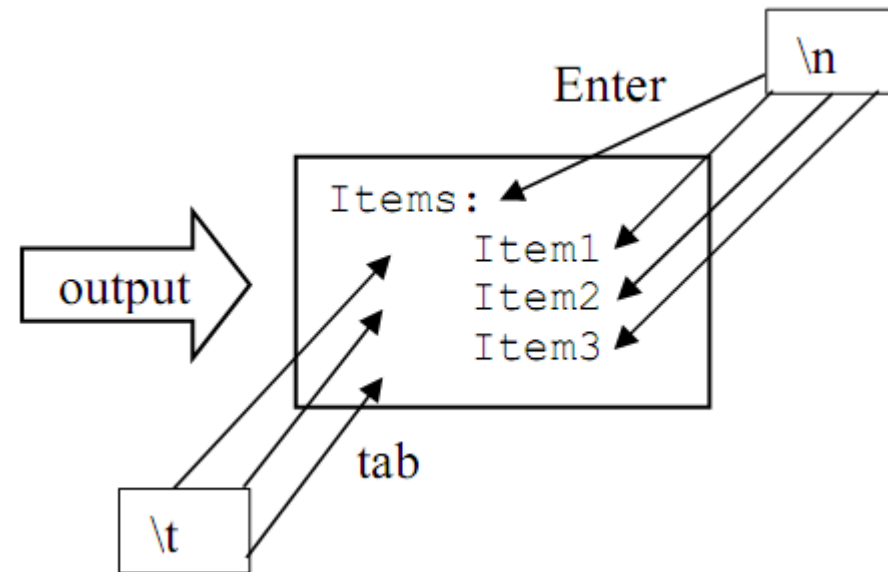
## 9. Hằng ký tự đặc biệt (escape sequences)

Mã	Ý nghĩa
\b	Lùi sang trái 1 ký tự
\f	Về đầu dòng
\n	Sang dòng mới
\r	Xuống dòng
\t	Tab theo chiều ngang
\"	Dấu nháy đôi
\'	Dấu nháy đơn
\0	Null
\\	Dấu \
\v	Tab theo chiều đứng
\a	Cảnh báo
\?	Dấu hỏi
\N	Hằng bát phân (với N là một hằng bát phân)
\xN	Hằng thập lục phân (với N là một hằng thập lục phân)

## 9. Hằng ký tự đặc biệt (escape sequences)



```
#include <iostream.h>
void main(void)
{
    cout << "Items:\n";
    cout << "\tItem1\n";
    cout << "\tItem2\n";
    cout << "\tItem3\n";
}
```







## 10. Toán tử (operators)

---

### ❑ Toán tử gán (assignment operator)

Cú pháp tổng quát

**`variableName = expression;`**

- *variableName*: Tên biến
- *expression*: Biểu thức

Lưu ý: phía bên trái dấu = phải là một **biến** hay **con trỏ** và không thể là hàm hay hằng.

Ví dụ:

`total = a + b + c + d;`

# 11. Chuyển đổi kiểu trong câu lệnh gán



1. Đối với câu lệnh gán, giá trị của biểu thức bên phải dấu = được tự động chuyển thành kiểu dữ liệu của biến bên trái dấu =

Ví dụ:

```
int i=100;
```

```
double d = 123.456;
```

1. Nếu thực thi lệnh **i = d**; thì **i = 123** (chuyển đổi kiểu mất mát thông tin).
2. Nếu thực thi lệnh **d = i**; thì **d = 100.0** (chuyển đổi kiểu không mất mát thông tin).

# 11. Chuyển đổi kiểu trong câu lệnh gán

---



1. Khi chuyển đổi từ kiểu dữ liệu có miền giá trị nhỏ sang kiểu dữ liệu có miền giá trị lớn hơn: `char`  $\rightarrow$  `int`  $\rightarrow$  `long`  $\rightarrow$  `float`  $\rightarrow$  `double`, thì việc chuyển đổi kiểu này là không mất mát thông tin
2. Khi chuyển đổi từ kiểu dữ liệu có miền giá trị lớn sang kiểu dữ liệu có miền giá trị nhỏ hơn: `double`  $\rightarrow$  `float`  $\rightarrow$  `long`  $\rightarrow$  `int`  $\rightarrow$  `char`, thì việc chuyển đổi kiểu này là mất mát thông tin



## 12. Toán tử số học (arithmetic operators)

Toán tử	Tên	Ví dụ
+	Cộng	12 + 4.9 // kết quả 16.9
-	Trừ	3.98 - 4 // kết quả -0.02
*	Nhân	2 * 3.4 // kết quả 6.8
/	Chia	9 / 2.0 // kết quả 4.5
%	Lấy phần dư	13 % 3 // kết quả 1



## 12. Toán tử số học (arithmetic operators)

---

1. Khi tử số và mẫu số của phép chia là số nguyên thì đó là phép chia nguyên nên phần dư của phép chia nguyên bị cắt bỏ.

Ví dụ:  $5/2$  cho kết quả là 2.

2. Toán tử lấy phần dư % (modulus operator) chỉ áp dụng với số nguyên.



## 13. Toán tử gán phức hợp

Toán Tử	Ví dụ	Tương đương với
<code>+=</code>	<code>n += 25</code>	<code>n = n + 25</code>
<code>-=</code>	<code>n -= 25</code>	<code>n = n - 25</code>
<code>*=</code>	<code>n *= 25</code>	<code>n = n * 25</code>
<code>/=</code>	<code>n /= 25</code>	<code>n = n / 25</code>
<code>%=</code>	<code>n %= 25</code>	<code>n = n % 25</code>



## 13. Toán tử gán phức hợp

---

```
#include <iostream.h>

int main ()
{
    int a, b=3;
    a = b;
    a+=2;          // tương đương với a=a+2
    cout << a;
    return 0;
}
```

## 14. Toán tử ++ và -- (increment and decrement operators)

---



1. Toán tử tăng (++) và toán tử giảm (--) có tác dụng làm tăng hoặc giảm 1 giá trị lưu trong biến.

2. Ví dụ:

`a++;` // tương đương với `a+=1;` và `a=a+1`

`a--;` // tương đương với `a-=1;` và `a=a-1`



## 14. Toán tử ++ và -- (increment and decrement operators)

---



Toán tử tăng/giảm có 2 dạng:

**1. Tiền tố (prefix):** Toán tử ++/-- đặt trước toán hạng, hành động tăng/giảm trên toán hạng được thực hiện trước, sau đó giá trị mới của toán hạng sẽ tham gia định trị của biểu thức.

2. Ví dụ:

B=3;

A=++B;

Kết quả: *A chứa giá trị 4, B chứa giá trị 4*

## 14. Toán tử ++ và -- (increment and decrement operators)

---



**1. Hậu tố (postfix):** Toán tử ++/-- đặt sau toán hạng, giá trị trong toán hạng được tăng/giảm sau khi đã tính toán.

2. Ví dụ:

B=3;

A=B++;

Kết quả: A chứa giá trị 3, B chứa giá trị 4

## 14. Toán tử ++ và -- (increment and decrement operators)

---



Ví dụ:

```
int x = 100;
```

```
int n,m;
```

```
n = ++x + 1; // n sẽ có giá trị là 102 (1)
```

```
n = x++ + 1; // n sẽ có giá trị là 101 (2)
```

1. Sau lệnh (1), (2) thì x có giá trị là 101

```
m = --x + 1; // m sẽ có giá trị là 100 (3)
```

```
m = x-- + 1; // m sẽ có giá trị 101 (4)
```

2. Sau lệnh (3), (4) thì x có giá trị là 99

## 14. Toán tử ++ và -- (increment and decrement operators)



1. Khi các toán tử số học xuất hiện trong một biểu thức, thì độ ưu tiên thực hiện như sau:

Toán tử	Độ ưu tiên
++   --	1
– (dấu âm)	2
*   /   %	3
+   –	4

# 15. Toán tử quan hệ & luận lý (relational & logical operators)



1. Toán tử quan hệ được định trị là true hoặc false.

Toán tử	Tên	Ví dụ
==	So sánh bằng	5 == 5 // kết quả 1
!=	So sánh không bằng	5 != 5 // kết quả 0
<	So sánh nhỏ hơn	5 < 5.5 // kết quả 1
<=	So sánh nhỏ hơn hoặc bằng	5 <= 5 // kết quả 1
>	So sánh lớn hơn	5 > 5.5 // kết quả 0
>=	So sánh lớn hơn hoặc bằng	6.3 >= 5 //kết quả1

# 15. Toán tử quan hệ & luận lý (relational & logical operators)



## 1. Toán tử luận lý:

Operator	Action	Ví dụ
!	Not	!(5 == 5) // kết quả là 0
&&	and	5 < 6 && 6 < 6 // kết quả là 0
	or	5 < 6    6 < 5 // kết quả là 1

## 2. Bảng chân trị:

P	Q	P&&Q	P  Q	!P
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

# 15. Toán tử quan hệ & luận lý (relational & logical operators)



## 1. Độ ưu tiên của toán tử quan hệ và luận lý:

Toán tử	Độ ưu tiên
!	1
> >= < <=	2
== !=	3
&&	4
	5

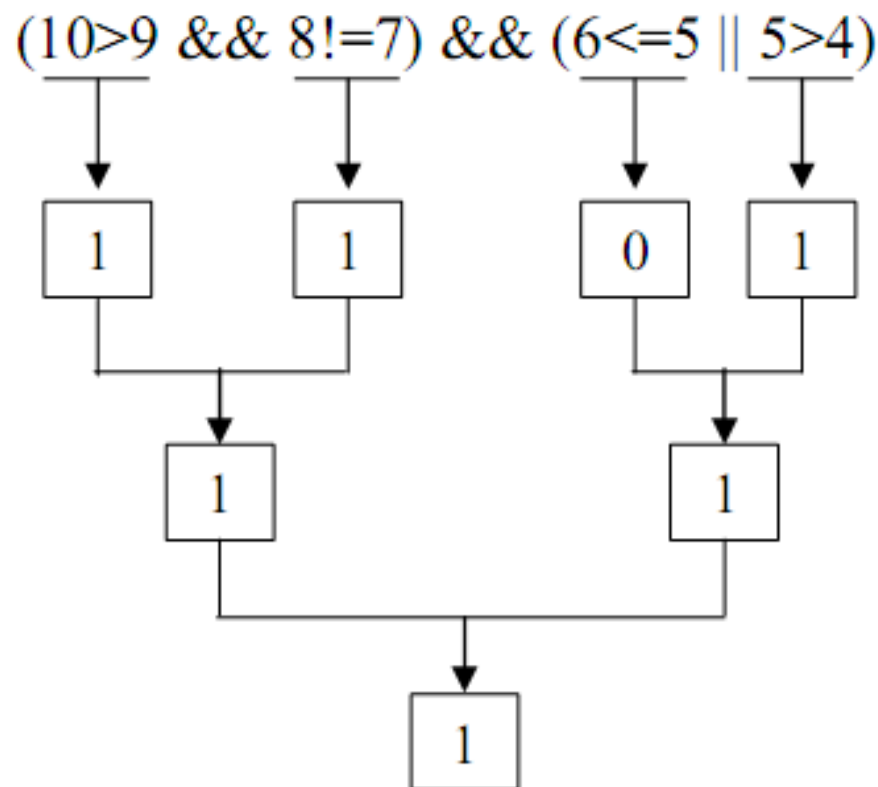
# 15. Toán tử quan hệ & luận lý (relational & logical operators)



Ví dụ biểu thức:

$(10 > 9 \ \&\& \ 8 \neq 7) \ \&\& \ (6 \leq 5 \ || \ 5 > 4)$

Được định trị như sau:







## 16. Toán tử ? (? operator)

---

1. Toán tử **?** là một toán tử ba ngôi do đó phải có ba toán hạng.
2. Dạng tổng quát của toán tử ? là:

**Exp1 ? Exp2 : Exp3;**

1. Exp1, Exp2, và Exp3 là các biểu thức.
2. Ý nghĩa:
  - Nếu Exp1 đúng thì Exp2 được định trị và nó trở thành giá trị của biểu thức.
  - Ngược lại, nếu Exp1 sai, Exp3 được định trị và trở thành giá trị của biểu thức.



## 16. Toán tử ? (? operator)

---

Ví dụ:

$X = 10$

$Y = X > 9 ? 100 * X : 200 * X$

Vì  $X > 9$  là true nên giá trị của biểu thức sẽ là 1000. Vậy y sẽ có giá trị là 1000.

Ví dụ:

$\text{int } m = 1, n = 2, p = 3;$

$\text{int min} = (\underline{m < n} ? (\underline{m < p} ? m : p) : (\underline{n < p} ? n : p));$



## 17. Toán tử sizeof

---

1. **sizeof** là toán tử một ngôi mà trả về số byte của kiểu dữ liệu chiếm trong bộ nhớ. Tùy môi trường (hệ điều hành, loại CPU,...) mà mỗi kiểu dữ liệu có số byte khác nhau.

2. Cú pháp:

**sizeof(operand)**

1. *operand*: có thể là tên kiểu dữ liệu, biến, biểu thức.

## 18. Toán tử dấu phẩy (comma operator)

---



1. Toán tử comma buộc các biểu thức cùng với nhau.
2. Biểu thức bên trái của toán tử comma luôn luôn được định trị như void, biểu thức bên phải được định trị và trở thành giá trị của biểu thức.
3. Dạng tổng quát của toán tử comma:

**(exp\_1, exp\_2, ..., exp\_n)**



## 18. Toán tử dấu phẩy (comma operator)

---

1. Các biểu thức được định trị từ trái sang phải, biểu thức cuối cùng ( $\text{exp\_n}$ ) được định trị và trở thành giá trị của toàn bộ biểu thức.
2. Ví dụ:

$x = (y=3, y+1);$

Y được gán giá trị 3, sau đó x được gán giá trị  $y+1$  là 4.



# 19. Độ ưu tiên của các toán tử

Cao nhất	( ) [ ] -> .
	! ~ ++ -- (type) * & sizeof
	* / %
	+ -
	<< >>
	< <= > >=
	== !=
	&
	^
	&&
	? :
	= += -= *= /= %=
Thấp nhất	,



## 20. Biểu thức (expressions)

---

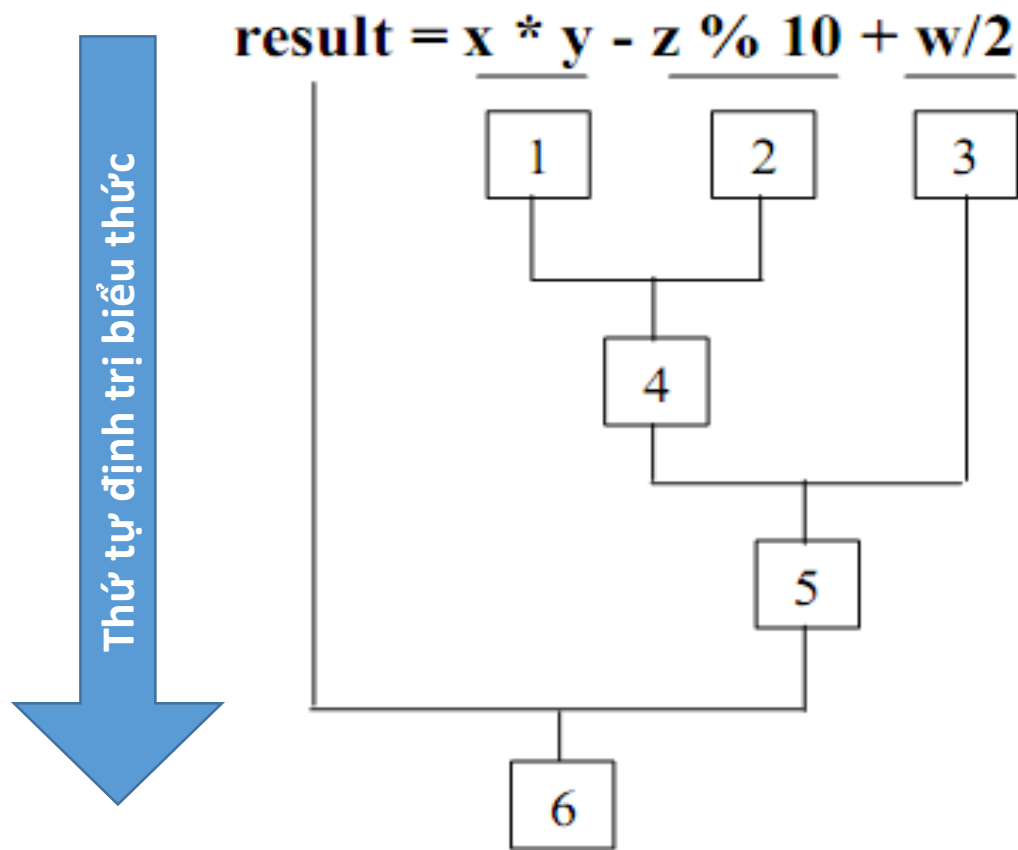
1. Một biểu thức trong C/C++ là sự kết hợp của các thành phần: toán tử, hằng, biến, và hàm có trả về giá trị.
2. Thứ tự định trị của biểu thức tùy thuộc vào độ ưu tiên của các toán tử.
3. Để biểu thức rõ ràng và thực hiện việc định trị đúng, nên dùng cặp dấu ngoặc tròn () bao quanh các biểu thức con của biểu thức.



## 20. Biểu thức (expressions)

1. Ví dụ: định trị biểu thức sau:

$$\text{result} = x * y - z \% 10 + w/2;$$







## 21. Chuyển kiểu trong biểu thức

---

1. Khi các hằng và biến của những kiểu khác nhau tồn tại trong một biểu thức, giá trị của chúng phải được chuyển thành cùng kiểu trước khi các phép toán giữa chúng được thực hiện.
2. Trình biên dịch sẽ thực hiện việc chuyển kiểu (convert) tự động đến kiểu của toán hạng có kiểu lớn nhất. Việc chuyển kiểu này gọi là thăng cấp kiểu (type promotion).



## 21. Chuyển kiểu trong biểu thức

1. Ví dụ:

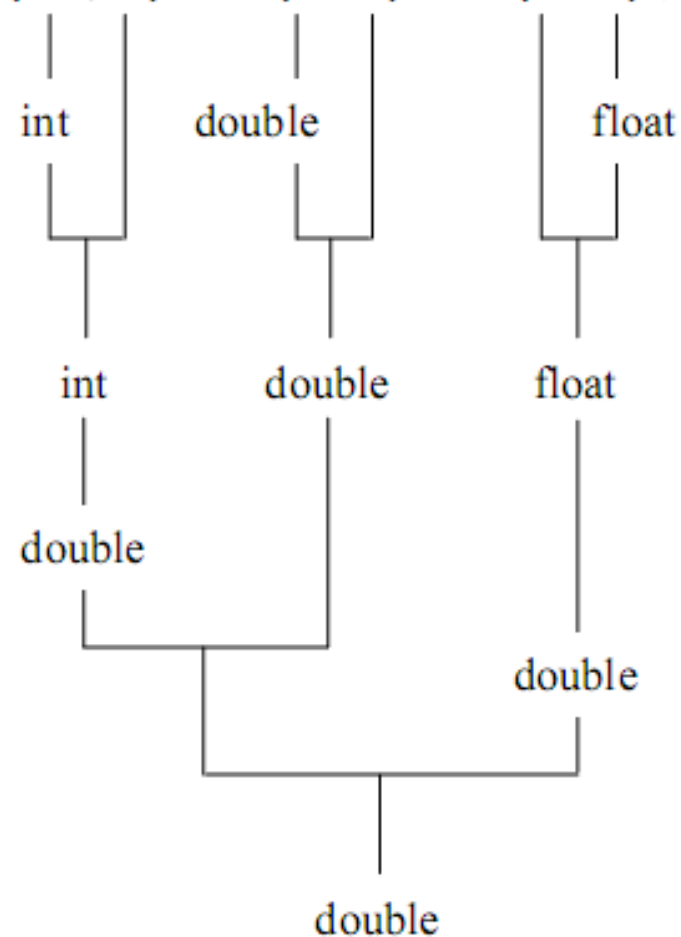
char ch;

int i;

float f;

double d;

```
result = (ch/i) + (f*d) - (f+i);
```





## 22. Ép kiểu (casting)

---

1. Casting dùng để ép kiểu của một biểu thức thành một kiểu theo ý muốn của lập trình viên.
2. Dạng tổng quát của casting là

**(type)expression**

Hoặc **type(expression)**

*type*: là tên một kiểu dữ liệu hợp lệ.

## 22. Ép kiểu (casting)

---



Ví dụ:

`float result;`

`result = 7/2;`

1. Do  $7/2$  là phép chia nguyên nên kết quả không có phần thập phân.
2. Sau lệnh trên `result` có giá trị là 3. Để phép chia trên là phép chia số thực ta thực hiện ép kiểu từ số hoặc mẫu số hoặc cả hai.



## 22. Ép kiểu (casting)

---

1. Ví dụ: Các cách viết sau đây cho cùng kết quả:

```
result = (float)7/2;
```

```
result = 7/(float)2;
```

```
result = (float)7/(float)2;
```

```
result = float(7)/float(2);
```



---

# **Nhập môn lập trình**

## **Bài 2- Các Câu Lệnh Rẽ Nhánh**

---



# Mục tiêu

---

- Hiểu và cài đặt được câu lệnh **if**
- Hiểu và cài đặt được câu lệnh **if...else**
- Hiểu và cài đặt được câu lệnh **if lồng nhau**
- Hiểu và cài đặt được câu lệnh **switch**

# 1. Giới thiệu

---



Có 3 loại cấu trúc điều khiển, Các cấu trúc này điều khiển thứ tự thực thi các lệnh của chương trình.

- ❑ **Cấu trúc tuần tự** (sequence): thực hiện các lệnh theo thứ tự từ trên xuống .
- ❑ **Cấu trúc lựa chọn** (selection): dựa vào kết quả của biểu thức điều kiện mà những lệnh tương ứng sẽ được thực hiện. Các cấu trúc lựa chọn gồm:
  - **If**
  - **switch.**



# 1. Giới thiệu

---



❑ **Cấu trúc lặp** (repetition or loop): lặp lại 1 hay nhiều lệnh cho đến khi biểu thức điều kiện có giá trị sai. Các cấu trúc lặp gồm:

- **for**
- **while**
- **do ... while.**

Tuy nhiên, thứ tự thực hiện các lệnh của chương trình còn bị chi phối bởi các lệnh nhảy như continue, break, goto.



## 2. Lệnh và khối lệnh

---

□ **Lệnh (*statement*):** một biểu thức kết thúc bởi 1 dấu chấm phẩy gọi là 1 lệnh.

Ví dụ:

```
int a, b, c ;  
a=10 ;  
a++;
```



## 2. Lệnh và khối lệnh

---

□ **Khối lệnh (*block*):** một hay nhiều lệnh được bao quanh bởi cặp dấu { } gọi là một khối lệnh. Về mặt cú pháp, khối lệnh tương đương 1 câu lệnh đơn.

Ví dụ:

```
if (a<b)
{
    temp=a;
    a=b;
    b=temp;
}
```



# Các cấu trúc lựa chọn



### 3. Cấu trúc IF



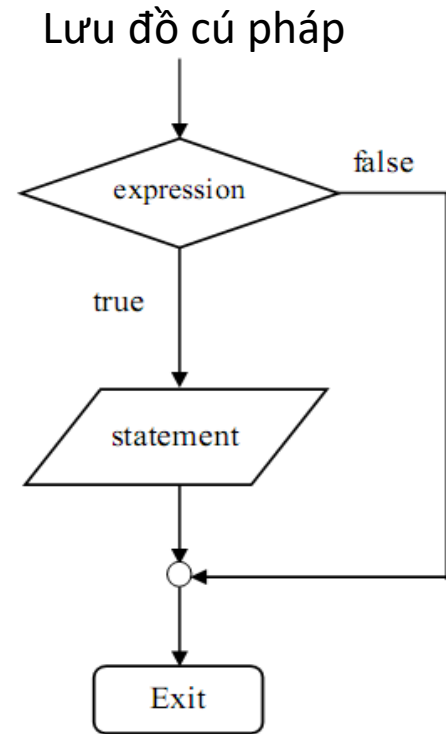
#### 1. Dạng 1:

##### 1. Cú pháp:

**if(expression)**  
**statement;**

##### ● Ý nghĩa:

**Expression** được định trị. Nếu kết quả là **true** thì **statement** được thực hiện, ngược lại, không làm gì cả.





### 3. Cấu trúc IF

---

Ví dụ: Viết chương trình nhập vào một số nguyên a. In ra màn hình kết quả a có phải là số dương không.

```
#include <iostream.h>
#include <conio.h>
int main()
{
    int a;
    cout << "Input a = "; cin>>a;
    if(a>=0)
        cout << a << " is a positive.";
    getch();
    return 0;
}
```

### 3. Cấu trúc IF



#### 1. Dạng 2:

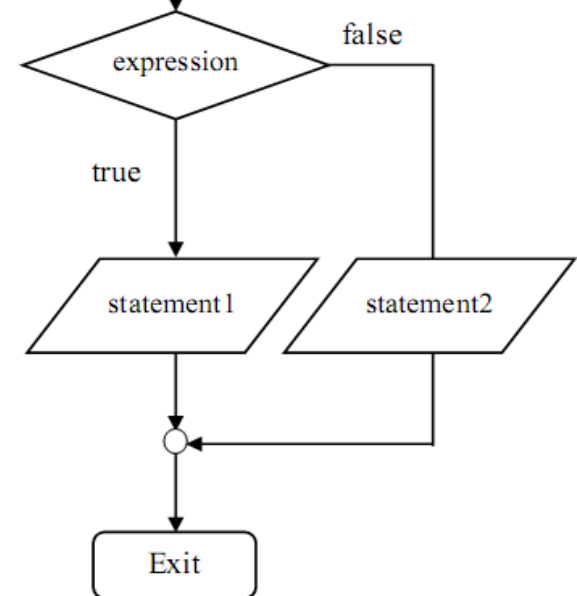
##### 1. Cú pháp:

```
if (expression)  
    statement1;  
else  
    statement2;
```

##### ● Ý nghĩa:

- Nếu **Expression** được định là **true** **statement1** được thực thi.
- Ngược lại, thì **statement2** được thực thi.

Lưu đồ cú pháp



### 3. Cấu trúc IF



Ví dụ: Viết chương trình nhập vào một số nguyên a. In ra màn hình kết quả kiểm tra a là số âm hay dương.

```
#include <iostream.h>
#include <conio.h>
int main()
{
    int a;
    cout << "Input a = "; cin >> a;
    if(a>=0)
        cout << a << " is a positive.";
    else
        cout << a << " is a negative.";
    getch(); return 0;
}
```



### 3. Cấu trúc IF

---



#### Lưu ý:

1. Ta có thể sử dụng các câu lệnh **if...else** lồng nhau. Khi dùng if...else lồng nhau thì else sẽ kết hợp với if gần nhất chưa có else.
2. Nếu câu lệnh if “bên trong” không có else thì phải đặt trong cặp dấu {}



## 4. Cấu trúc switch

---

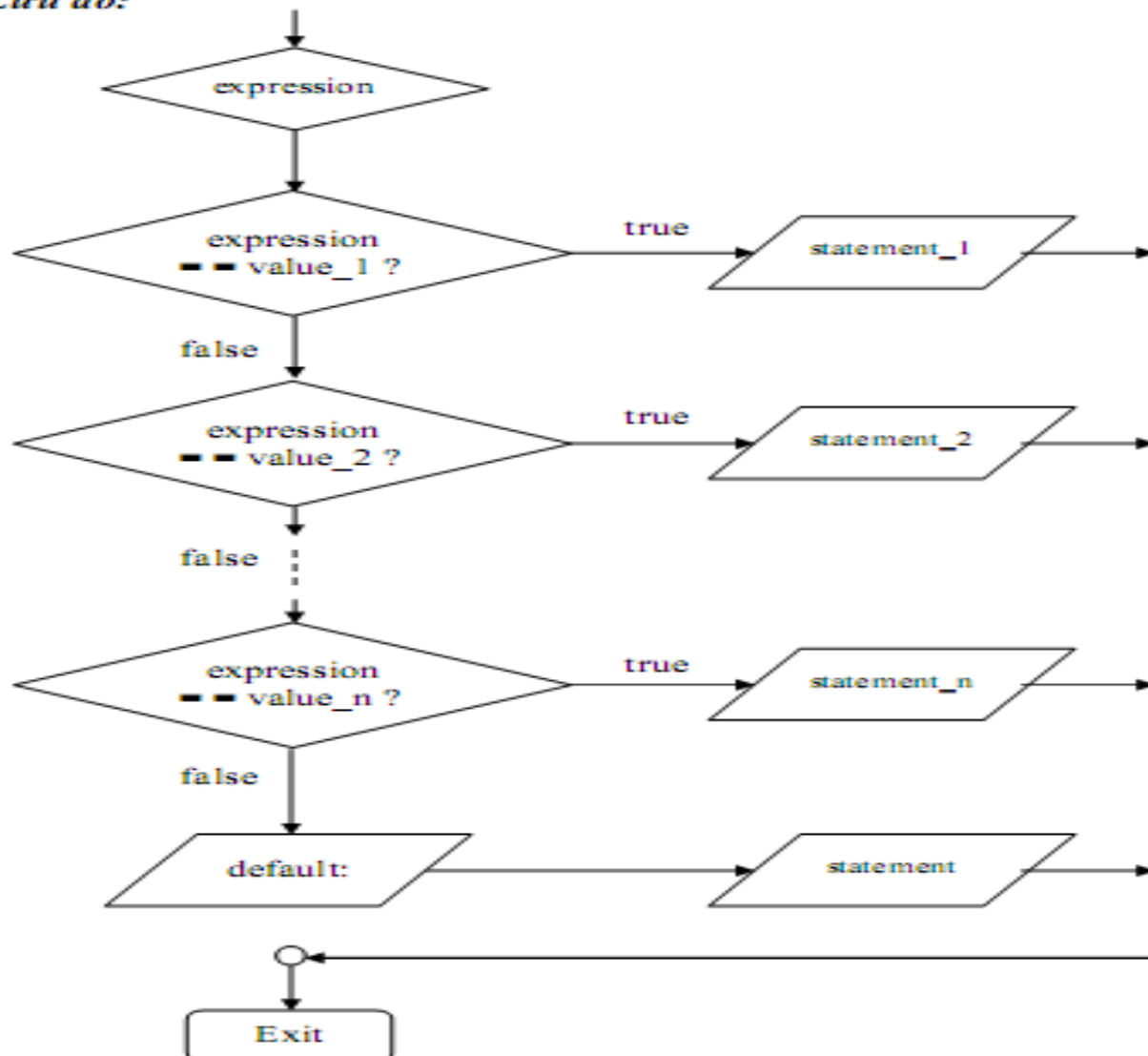
1. Cấu trúc switch là một cấu trúc lựa chọn có nhiều nhánh, được sử dụng khi có nhiều lựa chọn.
2. Cú pháp:

```
switch(expression)  
{  
    case value_1: statement_1; [break;]  
    ...  
    case value_n: statement_n; [break;]  
    [default : statement;]  
}
```

## 4. Cấu trúc switch



*Lưu đồ:*





## 4. Cấu trúc switch

---

### 1. Giải thích:

- Expression sẽ được định trị.
- Nếu giá trị của expression bằng value\_1 thì thực hiện statement\_1 và thoát.
- Nếu giá trị của expression khác value \_1 thì so sánh với value\_2, nếu bằng value\_2 thì thực hiện statement\_2 và thoát...., so sánh tới value\_n.
- Nếu tất cả các phép so sánh đều sai thì thực hiện statement của default.



## 4. Cấu trúc switch

---

### 1. Lưu ý:

- Expression trong switch() phải có kết quả là giá trị kiểu số nguyên (int, char, long).
- Các giá trị sau case phải là hằng nguyên.
- Không bắt buộc phải có default.
- Khi thực hiện lệnh tương ứng của case có giá trị bằng expression, chương trình thực hiện lệnh break để thoát khỏi cấu trúc switch.



## 4. Cấu trúc switch

---

Ví dụ: Nhập vào một số nguyên, chia số nguyên này cho 2 lấy phần dư. Kiểm tra nếu phần dư bằng 0 thì in ra thông báo “là số chẵn”, nếu số dư bằng 1 thì in thông báo “là số lẻ”.

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

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

```
void main () {
```

```
    int n, remainder;
```

```
    cout<<"Input an number: "; cin>>n; remainder = (n % 2);
```

```
    switch(remainder)
```

```
{
```

```
    case 0: cout << n << " is an even."; break;
```

```
    case 1: cout << n << " is an odd."; break;
```

```
}
```

```
    getch(); }
```



---

# Nhập môn lập trình

## Bài 3- Các Câu Lệnh Lặp

---



# Mục tiêu

---

- Hiểu và cài đặt được vòng lặp **for**
- Hiểu và cài đặt được vòng lặp **while**
- Hiểu và cài đặt được vòng lặp **do...while**
- Hiểu được cách sử dụng **continue, break**



# 1. Giới thiệu

---



❑ **Cấu trúc lặp** (repetition or loop): lặp lại 1 hay nhiều lệnh cho đến khi biểu thức điều kiện có giá trị sai. Các cấu trúc lặp gồm:

- **for**
- **while**
- **do ... while.**

Tuy nhiên, thứ tự thực hiện các lệnh của chương trình còn bị chi phối bởi các lệnh nhảy như **continue**, **break**, **goto**.

## 2. Cấu trúc for

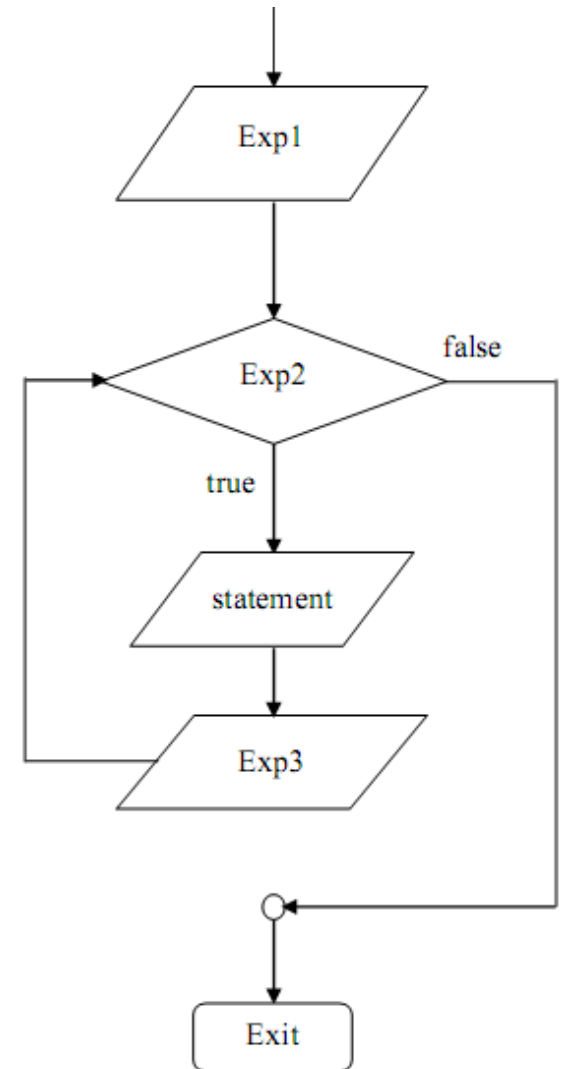


- Cú pháp:

**for (Exp1; Exp2; Exp3)  
statement;**

- Ý nghĩa:

- Exp1: là biểu thức khởi tạo được thực hiện.
- Exp2: là biểu thức điều kiện
- Exp3: biểu thức điều khiển lặp





## 2. Cấu trúc for

---

Ví dụ: Viết chương trình tính tổng các số nguyên từ 1 đến n.

```
#include <iostream.h>
#include <conio.h>
void main()
{
    int i, n, sum;
    cout<<"Input a number:"; cin >> n;
    sum = 0;
    for (i=1 ; i<=n ; i++)
        sum += i;
    cout<<"Sum from 1 to " << n << " is: " << sum;
    getch();
}
```



## 2. Cấu trúc for

---

1. C/C++ cho phép Exp1 là một định nghĩa biến

Ví dụ: `for(int i=1; i<=n; ++i)`

2. Bất kỳ biểu thức nào trong 3 biểu thức của vòng lặp for đều có thể rỗng

Ví dụ: `for(; i != 0;) statement;`

3. Xóa tất cả các biểu thức trong vòng lặp for sẽ cho một vòng lặp vô tận.

Ví dụ:

`for (;;) statement;`

# 3. Cấu trúc while



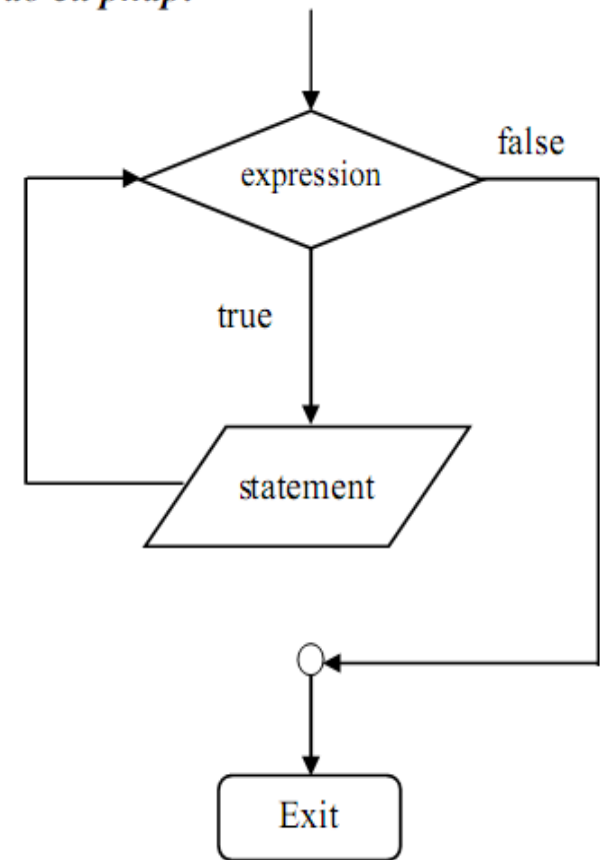
- Cú pháp:

**while(expression)**  
**statement;**

- Ý nghĩa:

- B1: Expression được định trị
- B2: Nếu kết quả là **true** thì statement thực thi và quay lại B1
- B3: Nếu kết quả là **false** thì thoát khỏi vòng lặp while.

*Lưu đồ cú pháp:*





### 3. Cấu trúc while

---

Ví dụ: Viết chương trình tính tổng các số nguyên từ 1 tới n.

```
#include<iostream.h>
#include<conio.h>
void main ()
{
    int i, n, sum;
    cout<<"Input n= "; cin >> n;
    i = 1; sum = 0;
    while(i<=n)
    {
        sum += i; i++;
    }
    getch();
}
```

# 4. Cấu trúc do ... while

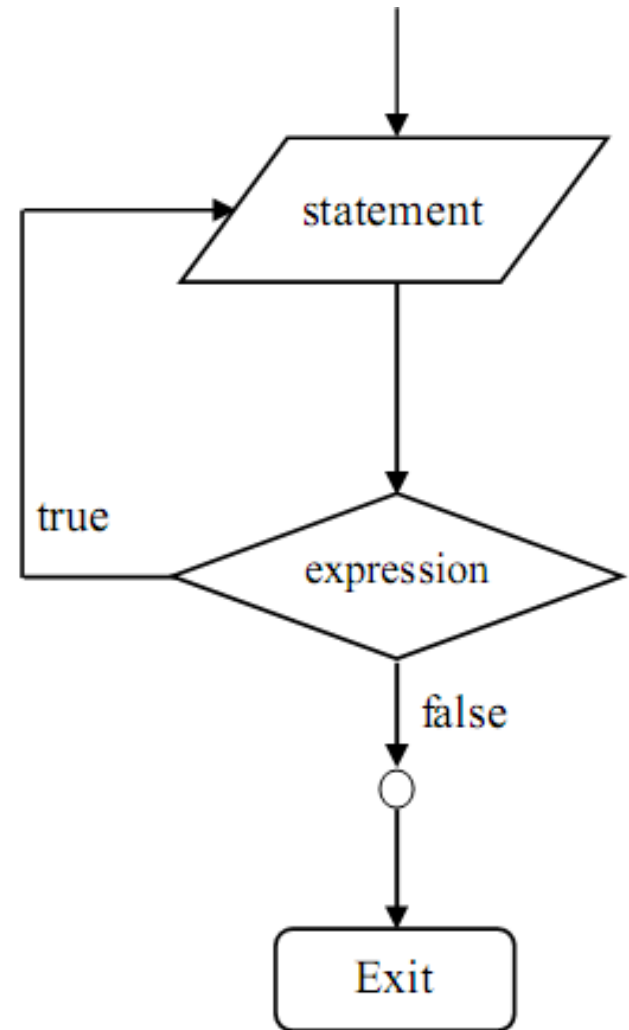


- Cú pháp:

```
do {  
    statement;  
}while(expression);
```

- Ý nghĩa:

- Statement được thực hiện
- Expression được định trị.
- Nếu expression là true thì quay lại bước 1
- Nếu expression là false thì thoát khỏi vòng lặp.





## 4. Cấu trúc do ... while

Ví dụ 1: Viết chương trình in dãy số nguyên từ 1 đến 10.

```
#include <iostream.h>
#include <conio.h>
void main ()
{
    int i;
    clrscr();
    cout<<"Display one to ten: ";
    i=1;
    do
    {
        cout << setw(3) << i;
        i+=1;
    } while(i<=10);
    getch();
}
```





# Các lệnh rẽ nhánh và lệnh nhảy





## 5. Lệnh break

---

1. Lệnh **break** dùng để thoát khỏi một cấu trúc điều khiển mà không chờ đến biểu thức điều kiện được định trị.
2. Khi **break** được thực hiện bên trong 1 cấu trúc lặp, điều khiển (control flow) tự động nhảy đến lệnh đầu tiên ngay sau cấu trúc lặp đó.
3. Không sử dụng lệnh **break** bên ngoài các cấu trúc lặp như while, do...while, for hay cấu trúc switch.



## 5. Lệnh break

---

Ví dụ: Đọc vào một mật khẩu người dùng tối đa attempts lần

```
for (i=0; i<attempts ; ++i)
{
    cout<<"Input a password: ";cin >> passWord;
    if (check(passWord)) //kiểm tra mật khẩu
        break;    // thoát khỏi vòng lặp
    cout <<"Password is wrong!\n";
}
```



## 6. Lệnh continue

---

1. Lệnh continue dùng để kết thúc vòng lặp hiện tại và bắt đầu vòng lặp tiếp theo.
2. Lệnh continue chỉ được dùng trong thân các cấu trúc lặp như for, while, do...while.
3. Câu lệnh continue thường đi kèm với câu lệnh if.



## 6. Lệnh continue

---

Ví dụ: Một vòng lặp thực hiện đọc một số, xử lý nó nhưng bỏ qua những số âm, và dừng khi số nhập vào là số 0.

```
do
{
    cin >> num;
    if (num < 0) continue;
    // process num here
} while(num != 0);
```



---

# Nhập môn lập trình

## Bài 4- Mảng

---



# Mục tiêu

---

- Hiểu và cài đặt được mảng
- Thao tác được trên mảng

M[0]	M[1]	M[2]	M[3]	M[4]	M[5]	M[6]
5	2	9	7	6	0	8



# 1. Khái niệm

---

- ✓ Mảng là một tập hợp các biến có cùng kiểu dữ liệu nằm liên tiếp nhau trong bộ nhớ và được tham chiếu bởi một tên chung chính là tên mảng.
- ✓ Mỗi phần tử của mảng được tham chiếu thông qua chỉ mục (index).

0	1	2	3	4	5	6	7	8	9
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]





# 1. Khái niệm

---

- ✓ Nếu mảng có  $n$  phần tử thì phần tử đầu tiên có chỉ mục là **0** và phần tử cuối có chỉ mục là  **$n-1$** .
- ✓ Để tham chiếu đến một phần tử ta dùng tên mảng và chỉ mục của phần tử được đặt trong cặp dấu [].

Ví dụ: `a[0]`

- ❖ Số phần tử trong mảng được gọi là *kích thước* của mảng. luôn *cố định*, phải được *xác định trước* và không đổi trong suốt quá trình thực hiện chương trình.

## 2. Mảng một chiều



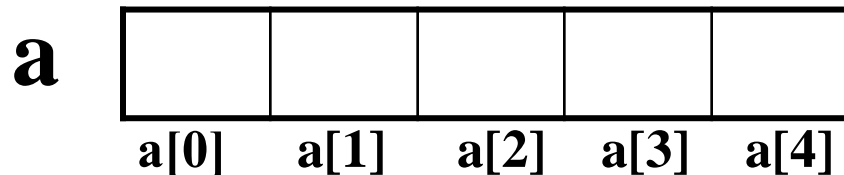
### Khai báo một mảng một chiều

✓ Cú pháp:

**type** **arrayName**[**elements**];

- **type**: kiểu dữ liệu của mỗi phần tử mảng.
- **elements**: số phần tử có trong mảng
- **arrayName**: tên mảng

● Ví dụ: `int a[5]`





## 2. Mảng một chiều

---

- ✓ Mảng phải được khai báo tường minh
- ✓ Kích thước (tính bằng byte) của mảng được tính theo công thức:

$$\text{Total\_size} = \text{sizeof}(\text{type}) * \text{elements}$$

Ví dụ:

```
int num[100];
```

Mảng num có kích thước là:

$2\text{bytes} * 100 = 200\text{bytes}$  (giả sử int chiếm 2 bytes)



## 2. Mảng một chiều

---

- ✓ Mỗi phần tử mảng là một biến thông thường.

Ví dụ:

```
int num[3];
```

```
num[0] = 2; //gán 2 cho phần tử num[0]
```

```
num[1] = num[0] + 3 //num[1] có giá trị 5
```

```
num[2] = num[0] + num[1]; //num[2] có giá trị 7
```

```
cout << num[1]; //In ra giá trị 5
```



## 2. Mảng một chiều

---

### *Khai báo và khởi tạo mảng một chiều*

✓ Cú pháp:

**type** **arrayName**[]= {value1, value2, ..., valuen};

✓ Lưu ý:

- Không khai báo kích thước mảng.
- Số lượng phần tử trong mảng là số các giá trị được cung cấp trong cặp dấu ngoặc {}, được phân cách nhau bởi dấu phẩy.



## 2. Mảng một chiều

---

✓ Ví dụ:

```
int soChan[] = {2,4,6,8,10};
```

Mảng **soChan** có 5 phần tử lần lượt là:

**soChan**[0] có giá trị là 2

**soChan**[1] có giá trị là 4

...

**soChan**[4] có giá trị là 10



## 2. Mảng một chiều

Ví dụ: Tạo một mảng nguyên a có N phần tử. Mỗi phần tử có giá trị là chỉ mục của nó. In mảng ra màn hình.

```
#include <iostream.h>
#include <conio.h>
#define N 10
void main()
{
    int a[N];
    for(int i=0 ; i < N ; i++)
        a[i] = i ;
    cout<< "In mang:\n";
    for(int i=0 ; i < N ; i++)
        cout << "a[" << i <<"] = " << a[i] <<
        endl;
}
```

## Ví dụ : Nhập vào một mảng số nguyên sau đó sắp xếp theo thứ tự tăng dần



```
#include <iostream.h>
#define n 5
int main ( )
{
    int a [ n ] ; int i , j , t ;
    for ( i = 0 ; i < n ; i ++ ) //nhập mảng
    { cout<<"a ["<<i<<" ] = " ; cin>>a[i]; cout<<endl; }
    for ( i = 0 ; i < n - 1 ; i ++ ) //sắp xếp
        for ( j = i + 1 ; j < n ; j ++ )
            if ( a [ i ] > a [ j ] )
                { t = a [ i ] ; a [ i ] = a [ j ] ; a [ j ] = t ; }
    for ( i = 0 ; i < n ; i ++ ) //xuất mảng
        cout<<setw(3)<<a[i];
    getch ( );
}
```



# Ví dụ: Đổi một số nguyên dương thập phân thành số nhị phân



```
void main()
```

```
{
```

```
    int i,j=0, n, np[20];
```

```
    cout<<"n="; cin>>n;
```

```
    do
```

```
    {
```

```
        np[j]= n%2;
```

```
        j++;
```

```
        n = n/2;
```

```
    }while(n>0);
```

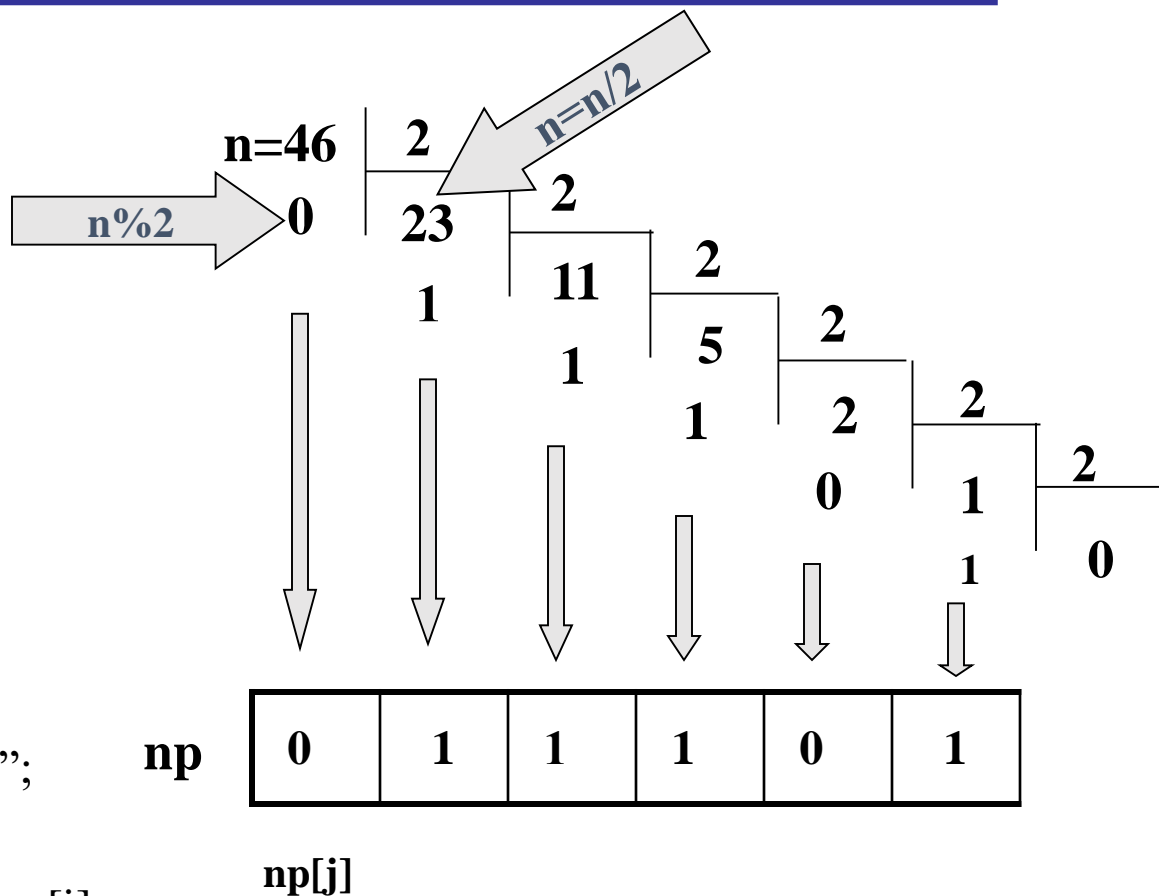
```
    cout<<"dang nhi phan: ";
```

```
    for(i=j-1 ; i>0 ; i--)
```

```
        cout<<setw(3)<<np[i];
```

```
    getch();
```

```
}
```





# Sử dụng hàm tạo số ngẫu nhiên

---

- ✓ C++ cung cấp hàm random để tạo ra các số ngẫu nhiên.
- ✓ Cú pháp:

**int random(int n)**

- ✓ Kết quả của hàm là tạo ra các số nguyên ngẫu nhiên từ 0 đến n-1
- ✓ Khi sử dụng random ta phải gọi *randomize* để khởi tạo chế độ tạo số ngẫu nhiên.
- ✓ Để sử dụng các hàm trên thì trong chương trình phải khai báo thư viện **<stdlib.h>**



## Ví dụ: tạo mảng ngẫu nhiên và in ra màn hình.

---

```
#include <iostream.h>
#include <conio.h>
#include <iomanip.h>
#include <stdlib.h>
void main()
{
    randomize();
    cout <<"Tao mang ngẫu nhiên :\n";
    for (int i=0; i<n; i++)
        a[i]=random(100);
    for(int j=0; j<n; j++)
        cout <<setw(3)<<a[j];
}
```



### 3. Mảng nhiều chiều

---

1. C/C++ hỗ trợ mảng nhiều chiều. Dạng đơn giản nhất của mảng nhiều chiều là mảng hai chiều.
2. Mảng hai chiều thực chất là mảng của những mảng một chiều. Ta có thể xem mảng hai chiều là một ma trận gồm các hàng và các cột

### 3. Mảng nhiều chiều



#### Khai báo mảng hai chiều

**type** arrayName[**rows**][**columns**];

- **rows**: số hàng
- **columns**: số cột

Ví dụ: Khai báo mảng số nguyên 3 hàng 4 cột

int a[3][4]

num [t] [i]

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12



### 3. Mảng nhiều chiều

---

1. *Khai báo và khởi tạo mảng hai chiều*

2. Cú pháp:

```
type arrayName[][columns] = {  
    {value1,value2,...,valueN},  
    {value1,value2,...,valueN},  
    {...},  
    {value1,value2,...,valueN}};
```



### 3. Mảng nhiều chiều

---

- ✓ Số phần tử của mỗi hàng phải bằng số cột
- ✓ Số hàng của khai báo mảng hai chiều để trống.
- ✓ Số hàng của mảng được xác định dựa vào số hàng trong phần khởi tạo. Giá trị các phần tử trong mỗi hàng được đặt trong cặp {}, các hàng phân cách nhau bằng một dấu phẩy.
- ✓ Ví dụ:

```
int a[][4] = {{1,2,3,4}, {5,6,7,8},{9,10,11,12}};
```



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

---

```
void main()
{
    int a[4][3];
    srand(time(NULL));
    for(int i=0 ; i<4 ; i++)
        for(int j=0 ; j<3 ; j++)
            a[i][j] = rand()%10;
    for(int i=0 ; i<4 ; i++)
    {
        for(int j=0 ; j<3 ; j++)
            cout << a[i][j] << " ";
        cout << endl;
    }
}
```



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

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

---

```
void main()
```

```
{
```

```
    int a[][4] = {{1,2,3,4},{5,6,7,8},{9,10,11,12},  
                  {13,14,15,16}};
```

```
    int sum=0;
```

```
    for(int i=0 ; i<4 ; i++)
```

```
        for(int j=0 ; j<4 ; j++)
```

```
            if(i==j)
```

```
                sum += a[i][j];
```

```
    cout << "Tong duong cheo chinh la: " << sum;
```

```
}
```



# KỸ THUẬT LẬP TRÌNH





## Bài 5:

# HÀM ( CHƯƠNG TRÌNH CON )



## **1. Cấu trúc và lý do sử dụng chương trình con**

### **2. Tham số cho chương trình con**

**Truyền tham số cho chương trình:  
tham trị, tham biến**

### **3. Chương trình đệ quy**

**Một số bài toán đệ quy thông thường**



# 1. Cấu trúc hàm và lý do sử dụng hàm



## 1.1. Khái niệm

- Hàm là một khối lệnh thực hiện một công việc hoàn chỉnh (module), được đặt tên và được gọi thực thi nhiều lần tại nhiều vị trí trong chương trình.
- Hàm còn gọi là chương trình con (*subroutine*)

## 1.1. Khái niệm

- Hàm có thể được gọi từ chương trình chính (hàm main) hoặc từ 1 hàm khác.
- Hàm có giá trị trả về hoặc không. Nếu hàm không có giá trị trả về gọi là thủ tục (*procedure*)

## 1.1. Khái niệm

- Có hai loại hàm:
  - *Hàm thư viện*: là những hàm đã được xây dựng sẵn. Muốn sử dụng các hàm thư viện phải khai báo thư viện chứa nó trong phần khai báo `#include`.
  - *Hàm do người dùng định nghĩa*.



## 1.2. Dạng tổng quát của hàm

- Dạng tổng quát của hàm do người dùng định nghĩa:

**returnType** **functionName**(parameterList)

{

**body of the function**

Kiểu dữ liệu

Tên hàm

Tham số

## 1.2. Dạng tổng quát của hàm

```
1 #include<iostream.h>
2 #include<conio.h>
3 int TonghaIso(int a,int b);
4 void main()
5 {
6     int c,d,kq;
7     cout<<"Nhap c = ";
8     cin>>c;
9     cout<<"Nhap d = ";
10    cin>>d;
11    kq=TonghaIso(c,d);
12    cout<<"Tong la: "<<kq;
13 }
14 int TonghaIso(int a,int b)
15 {
16     return a+b;
17 }
```

Gọi hàm

Truyền đối số

Tham số

## 1.2. Dạng tổng quát của hàm

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

```
int conghaiso(int a,int b);
```

```
void main()
```

```
{
```

```
}
```

```
int conghaiso(int a,int b)
```

```
{
```

```
    return a+b;
```

```
}
```

```
void ham1(int x,int y)
```

```
{
```

```
    return x+y; SAI
```

```
}
```

- Vậy từ khóa **return** có tác dụng gì trong hàm?
- Khi một hàm muốn trả về một giá trị nào đó thì chúng ta dùng return . Bất kỳ kiểu dữ liệu nào của hàm cũng có thể sử dụng return **NGOẠI TRỪ** kiểu **void**

Hàm có kiểu void đôi khi được gọi là Thủ Tục

### 1.3. Gọi hàm

- Một hàm khi đã định nghĩa nhưng chúng vẫn chưa được thực thi, hàm chỉ được thực thi khi trong chương trình có một lời gọi đến hàm đó.
- **Cú pháp gọi hàm:**

**<Tên hàm>([Danh sách các tham số])**

## 1.4. Nguyên tắc hoạt động của hàm

```
void main()
{
    int a, b, USC;
    cout<<"Nhap a,b: ";
    cin>>a>>b;
    USC = uscln(a,b);
    cout<<"Uoc chung lon
    nhat la: ", USC);
}
```

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



## 2. Tham số cho chương trình con



## 2.1. Tham số hình thức & tham số thực

➤ Khi hàm cần nhận đối số (*arguments*) để thực thi thì khi khai báo hàm cần khai báo danh sách các tham số để nhận giá trị từ chương trình gọi. Các tham số này được gọi là ***tham số hình thức***.

Ví dụ:

```
int min(int a, int b)
{
    if(a < b)
        return a;
    else
        return b;
}
```

Tham số hình thức

## 2.1. Tham số hình thức & tham số thực

- Khi gọi hàm, ta cung cấp các giá trị thật, các giá trị này sẽ được sao chép vào các tham số hình thức và các giá trị thật được gọi là ***tham số thực***.

Ví dụ: Để tìm giá trị nhỏ nhất của 2 số 5 và 6 ta gọi hàm **min(5, 6)**

Tham số thực

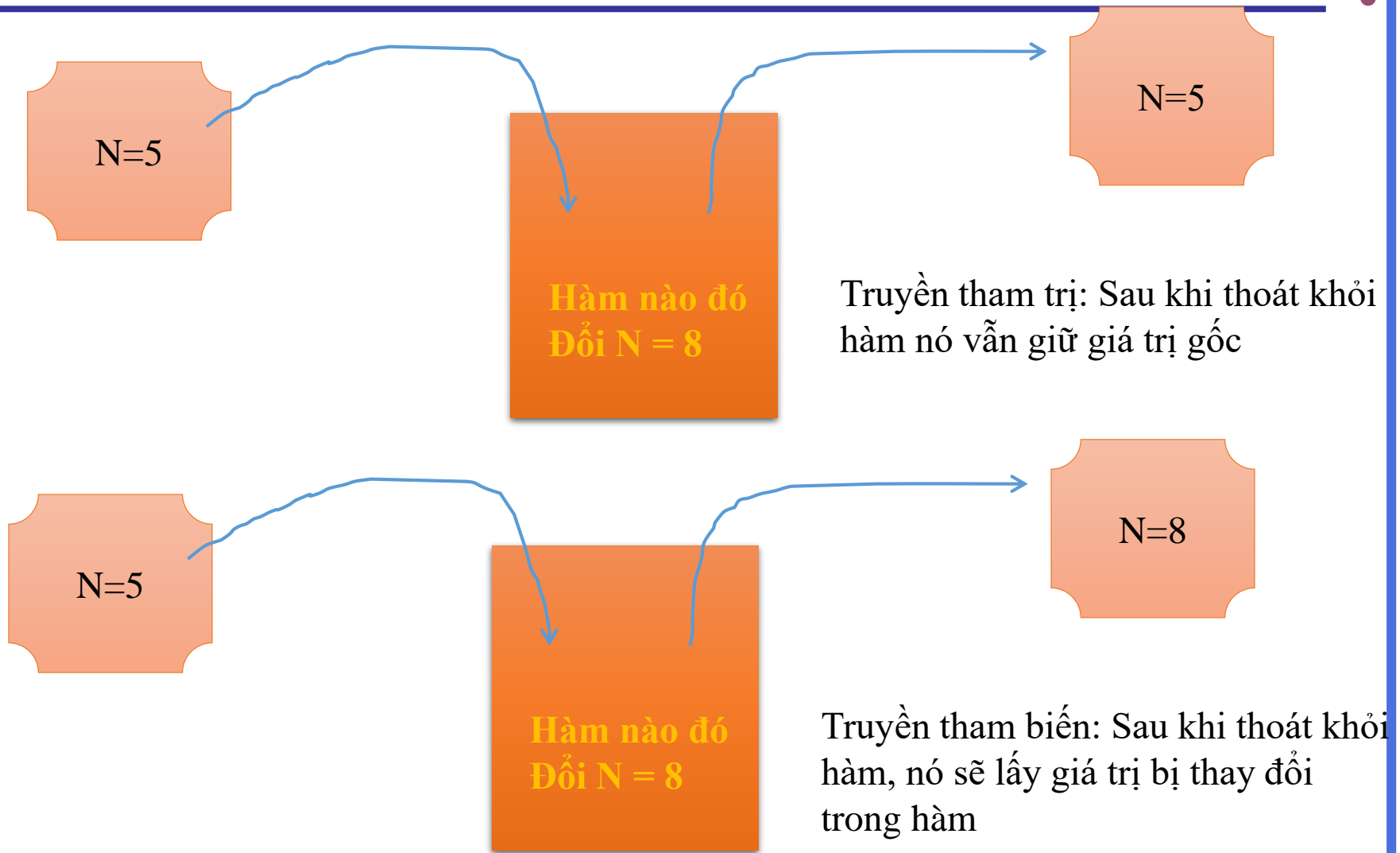
**min(int a, int b)**



## 2.1. Tham số hình thức & tham số thực

- Có hai cách truyền đối số vào tham số hình thức:
  - Truyền *tham trị*
  - Truyền *tham biến*.

## 2.1. Tham số hình thức & tham số thực



## 2.1. Tham số hình thức & tham số thực

- *Truyền tham trị (call by value)*
  - Sao chép giá trị của *đối số* vào *tham số hình thức* của hàm.
  - Những thay đổi của tham số không ảnh hưởng đến giá trị của đối số.

## 2.1. Tham số hình thức & tham số thực

Ví dụ:

```
void hamgido(int a)
{
    a = a*2;
    cout << "gia tri cua a
    trong ham double:" << a;
}
```

```
void main()
{
    int a=40;
    hamgido (a);
    cout << "\n Gia tri cua a
    trong ham main: ";
    cout << "a = " << a <<
    endl;
}
```

## 2.1. Tham số hình thức & tham số thực

```
void main()
{
    int a= 40
    hamgido (a);
    cout<<“\n Gia tri cua a
    trong ham main: ”;
    cout << “a = “ << a <<
    endl;
}
```

**Gia tri cua a trong ham hamgido: 80**  
**Gia tri cua a trong ham main: 40**

```
void hamgido ( int a )
{
    a = a *2;//
    cout << “Gia tri cua
    a trong ham
    double:“<< a;
}
```

## 2.1. Tham số hình thức & tham số thực

- **Truyền tham chiếu (*call by reference*)**
  - **Sao chép địa chỉ** của đối số vào tham số hình thức. Do đó, những thay đổi đối với tham số sẽ có tác dụng trên đối số.

Ví dụ: Khi gọi hàm hamgido (&a);

Địa chỉ của a truyền vào cho tham số hình thức của hàm: hamgido (int &b)

## 2.1. Tham số hình thức & tham số thực

```
void main()
{
    int a=40;
    hamgido (a);
    cout << "\Trong ham
    main : a = " << a ;
}
```

```
void hamgido ( int &b)
{
    b*= 2;
    cout << "Trong hàm
    double a = " << b;
}
```

**Trong hàm hamgido a = 80**  
**Trong hàm main a = 40**

## 2.1. Tham số hình thức & tham số thực

```
#include<iostream.h>
void ham1(int x);//truyen tham tri
void ham2(int &x);//truyen tham bien
void main()
{
    int x=5;
    cout<<"x ban dau = "<<x;
    cout<<"\n";
    ham1(x);
    cout<<"x sau khi goi ham1 = "<<x;
    cout<<"\n";
}
void ham1(int x)
{
    x=10;
    cout<<"x trong ham1 ="<< x <<"\n";
}

void ham2(int &x)
{
    x=10;
    cout<<x <<"\n";
}
```

**Gọi hàm truyền tham trị**

```
#include<iostream.h>
void ham1(int x);//truyen tham tri
void ham2(int &x);//truyen tham bien
void main()
{
    int x=5;
    cout<<"x ban dau = "<<x;
    cout<<"\n";
    ham2(x);
    cout<<"x sau khi goi ham2 = "<<x;
    cout<<"\n";
}
void ham1(int x)
{
    x=10;
    cout<<"x trong ham1 ="<< x <<"\n";
}

void ham2(int &x)
{
    x=10;
    cout<<x <<"\n";
}
```

**Gọi hàm truyền tham biến**



## 2.1. Prototype (nguyên mẫu) của hàm

- Chương trình **bắt buộc phải có prototype của hàm** hoặc phải **bắt buộc viết định nghĩa của hàm trước khi gọi**.
- Sau khi đã sử dụng prototype của hàm, ta có thể viết định nghĩa chi tiết hàm ở bất kỳ vị trí nào trong chương trình.

## 2.1. Prototype (nguyên mẫu) của hàm

```
#include <iostream.h> // Khai báo thư viện iostream.h
int max(int x, int y); // khai báo nguyên mẫu hàm max
void main() // hàm main (sẽ gọi các hàm thực hiện)
{
    int a, b; // khai báo biến
    cout<<" Nhập vào 2 số a, b ";
    cin>>a>>b;
    cout<<"số lớn nhất là:"<< max(a,b);
}
int max(int x, int y) // Định nghĩa hàm max(a,b)
{
    return (x>y) ? x:y;
}
```

### 3. đệ qui

- Một hàm được gọi là đệ qui nếu một lệnh trong thân hàm gọi đến chính hàm đó.
- Đệ qui giúp giải quyết bài toán theo cách nghĩ thông thường một cách tự nhiên.
- Đệ qui phải xác định được **điểm dừng**. Nếu không xác định chính xác thì làm bài toán bị sai và có thể bị lặp vĩnh cửu (Stack Overhead)

### 3. Độ qui

- Ví dụ: Định nghĩa giai thừa của một số nguyên dương  $n$  như sau:
- $5! = 5 * 4!$
- $4! = 4 * 3!$
- Tức là nếu ta biết được  $(n-1)$  giai thừa thì ta sẽ tính được  $n$  giai thừa, vì  $n! = n * (n-1)!$
- Thấy  $n=0$  hoặc  $n=1$  thì giai thừa luôn  $= 1 \rightarrow$  chính là điểm dừng

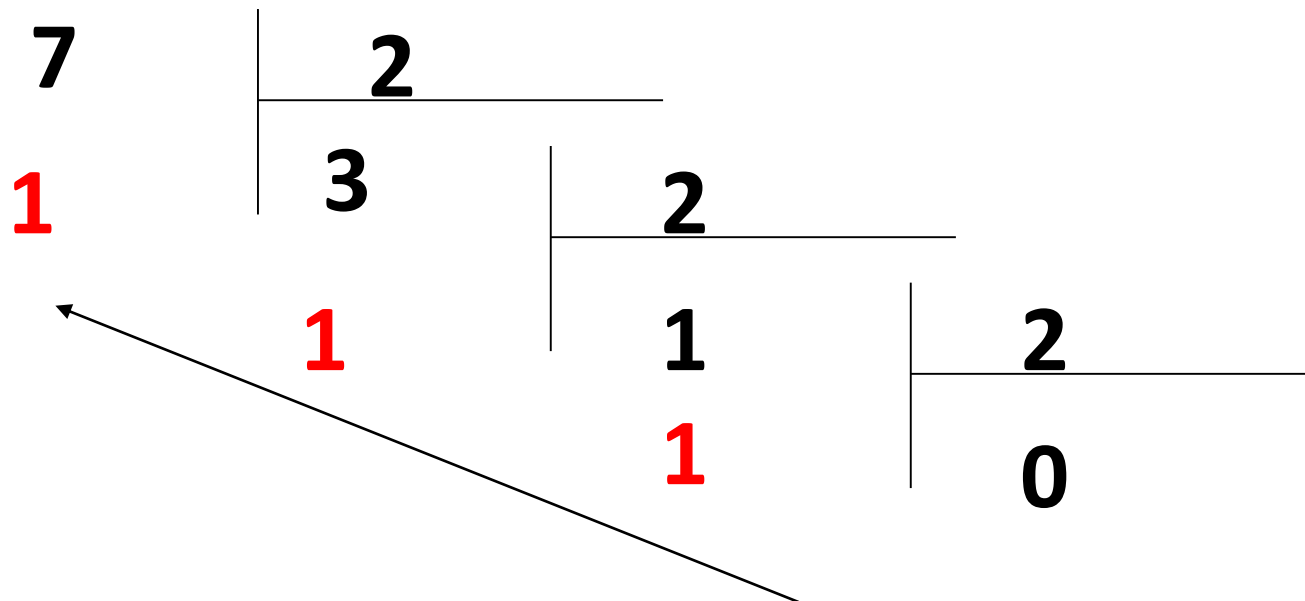
$$n! = 1 * 2 * 3 * \dots * (n-1) * n = (n-1)! * n \text{ (với } 0! = 1)$$

### 3. Đệ qui

```
int giaiThua(int n)
{
    if(n<=1)
        return(1);
    return n*giaiThua(n-1); // gọi đệ qui
}
```

### 3. Độ qui

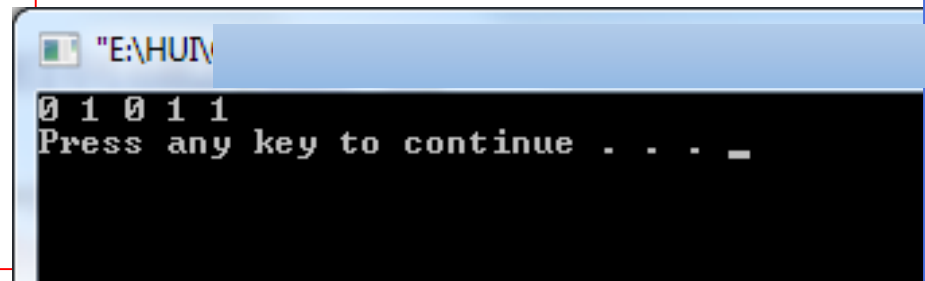
**Ví dụ : Dùng độ qui để chuyển đổi từ hệ thập phân sang nhị phân.**



### 3. Đệ qui

```
void H10toH2(int n)
{
    int t=n%2;
    if(n>0)
        H10toH2(n/2);
    cout<<t<<" ";
}
```

```
void main()
{
    H10toH2(11);
}
```



The screenshot shows a Windows command prompt window with the title bar "E:\HUN". The command prompt displays the output of the program: "0 1 0 1 1". Below the output, it says "Press any key to continue . . . \_".

### 3. Độ qui

#### **\*Phân loại độ qui**

- \*Độ qui tuyến tính.
- \*Độ qui nhị phân.
- \*Độ qui phi tuyến.
- \*Độ qui hồ tương.



### 3. Định nghĩa hàm

Trong thân hàm có duy nhất một lời gọi hàm gọi lại chính nó một cách tường minh.

<Kiểu dữ liệu hàm> **TenHàm** (<danh sách tham số>)

```
{  
if (điều kiện dừng)  
{  
...  
//Trả về giá trị hay kết thúc công việc  
}  
//Thực hiện một số công việc (nếu có)  
... TenHàm (<danh sách tham số>);  
//Thực hiện một số công việc (nếu có)  
}
```

### 3. Độ qui tuyến tính

#### Ví dụ:

Tính  $S(n) = 1 + 2 + 3 + \dots + n$

- Điều kiện dừng:  $S(0) = 0$ .

- Quy tắc (công thức) tính:  $S(n) = S(n-1) + n$ .

*long TongS (int n)*

{

*if(n==0)*

*return 0;*

*return ( TongS(n-1) + n );*

}

## 3.2. Độ qui nhị phân

Trong thân của hàm có hai lời gọi hàm gọi lại chính nó một cách tường minh.

<Kiểu dữ liệu hàm> **TenHam** (<danh sách tham số>)

{

    if (điều kiện dừng)

{

...

//Trả về giá trị hay kết thúc công việc

}

//Thực hiện một số công việc (nếu có)

...**TenHam** (<danh sách tham số>); //Giải quyết vấn đề nhỏ hơn

//Thực hiện một số công việc (nếu có)

... **TenHam** (<danh sách tham số>); //Giải quyết vấn đề còn lại

//Thực hiện một số công việc (nếu có)

}

## 3.2. Độ qui nhị phân

Ví dụ: Tính số hạng thứ  $n$  của dãy Fibonacci được định nghĩa như sau:

$$f_1 = f_0 = 1 ; \quad (n > 1)$$

$$f_n = f_{n-1} + f_{n-2} ;$$

*Điều kiện dừng:  $f(0) = f(1) = 1$ .*

```
long Fibonacci (int n)  
{  
    if(n==0 || n==1)  
        return 1;  
    return Fibonacci(n-1) + Fibonacci(n-2);  
}
```

### 3. 3. Độ qui phi tuyến

Trong thân của hàm có lời gọi hàm gọi lại chính nó được đặt bên trong vòng lặp.

<Kiểu dữ liệu hàm> **TenHam** (<danh sách tham số>)

```
{  
for (int i = 1; i<=n; i++)  
{  
//Thực hiện một số công việc (nếu có)  
    if (điều kiện dừng)  
    { ...  
//Trả về giá trị hay kết thúc công việc  
}  
else  
{ //Thực hiện một số công việc (nếu có)  
TenHam (<danh sách tham số>);  
}  
}  
}
```

### 3.3. Độ qui phi tuyến

Ví dụ: Tính số hạng thứ  $n$  của dãy  $\{X_n\}$  được định nghĩa như sau:

$$X_0 = 1 ;$$

$$X_n = n^2 X_0 + (n-1)^2 X_1 + \dots + 1^2 X_{n-1} ; \quad (n \geq 1)$$

*Điều kiện dừng:*  $X(0) = 1$ .

***long TinhXn (int n)***

***{***

***if(n==0)***

***return 1;***

***long s = 0;***

***for (int i=1; i<=n; i++)***

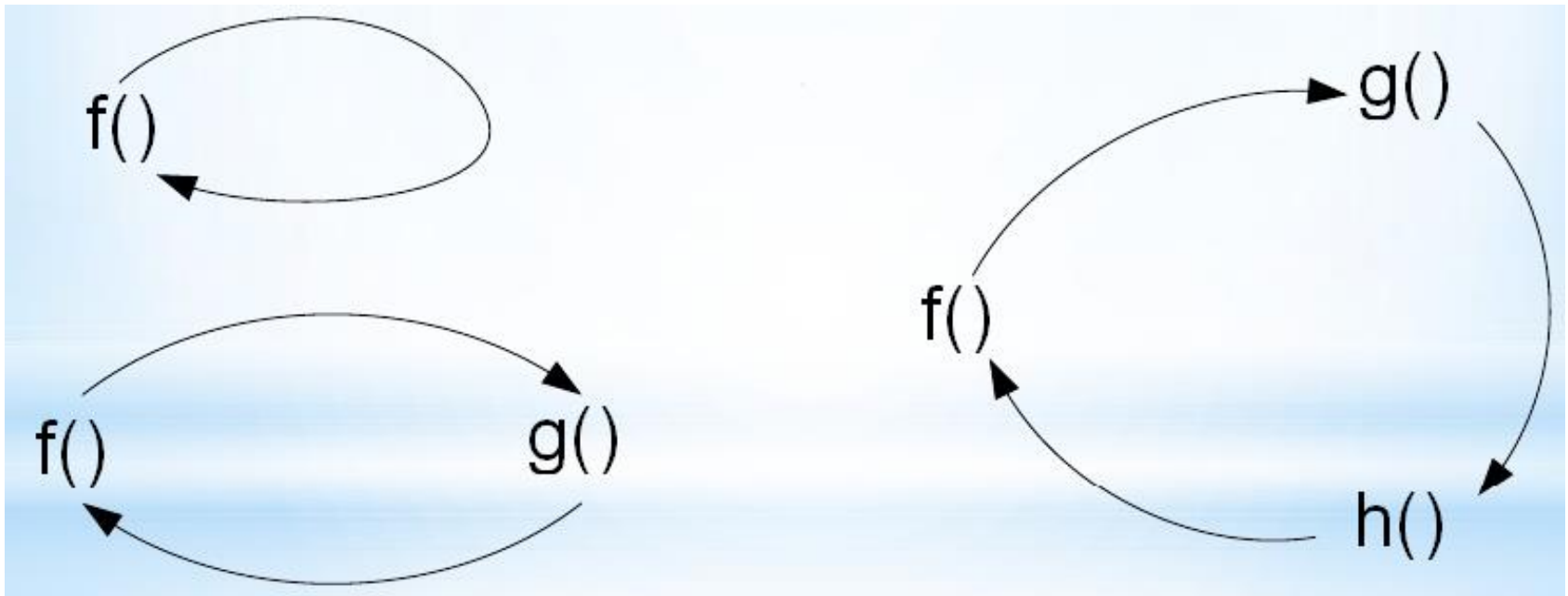
***s = s + i \* i \* TinhXn(n-i);***

***return s;***

***}***

### 3.3. Độ qui hỗ tương

Trong thân của hàm này có lời gọi hàm đến hàm kia và trong thân của hàm kia có lời gọi hàm tới hàm này.



### 3. 3. Độ qui hỗ tương

```
<Kiểu dữ liệu hàm> TenHam2 (<danh sách tham số>);  
<Kiểu dữ liệu hàm> TenHam1 (<danh sách tham số>)  
{  
    //Thực hiện một số công việc (nếu có)  
    ...TenHam2 (<danh sách tham số>);  
    //Thực hiện một số công việc (nếu có)  
}  
<Kiểu dữ liệu hàm> TenHam2 (<danh sách tham số>)  
{  
    //Thực hiện một số công việc (nếu có)  
    ...TenHam1 (<danh sách tham số>);  
    //Thực hiện một số công việc (nếu có)  
}
```



### 3.3. Đệ qui hỗ tương

Ví dụ: Tính số hạng thứ  $n$  của hai dãy  $\{X_n\}$ ,  $\{Y_n\}$  được định nghĩa như sau:

$$X_0 = Y_0 = 1;$$

$$X_n = X_{n-1} + Y_{n-1}; \quad (n > 0)$$

$$Y_n = n^2 X_{n-1} + Y_{n-1}; \quad (n > 0)$$

- Điều kiện dừng:  $X(0) = Y(0) = 1$ .

***long TinhYn(int n);***

***long TinhXn (int n)***

***{***

***if(n==0)***

***return 1;***

***return TinhXn(n-1) + TinhYn(n-1);***

***}***

***long TinhYn (int n)***

***{***

***if(n==0)***

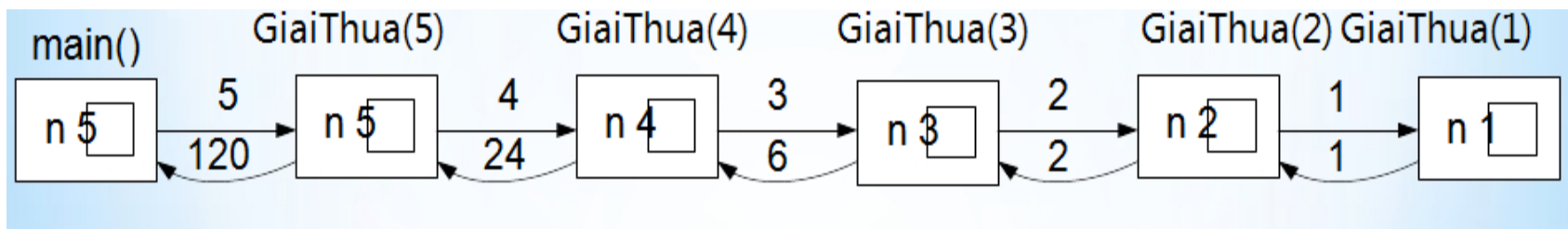
***return 1;***

***return n\*n\*TinhXn(n-1) + TinhYn(n-1);***

***}***

### 3. 4. Cách hoạt động của hàm đệ qui

Ví dụ tính  $n!$  với  $n=5$





## Bài 6:

# Xử Lý Chuỗi & Con trỏ





**1. Khái niệm và cấu trúc về chuỗi**

**2. Các hàm nhập xuất chuỗi**

**3. Một số hàm cơ bản về chuỗi**

**4. Mảng và chuỗi**

# 1. Khái niệm

- Chuỗi là một **mảng ký tự** được kết thúc bằng ký tự **null** ('\0').
- Ký tự **null** ('\0') là ký tự dùng để kết thúc Chuỗi
- **Hằng Chuỗi** là Chuỗi được bao quanh bởi cặp **dấu nháy đôi**. Ví dụ: "Hello"
- Ví dụ: để khai báo một mảng **str** chứa chuỗi có độ dài 20 ký tự, ta khai báo:

```
char str[21];
```

## 1. 1. Khai báo và khởi tạo Chuỗi

- ❖ Có 2 cách khai báo và khởi tạo Chuỗi
  - Cách 1: Dùng mảng một chiều

**char <Tên biến> [Chiều dài tối đa]**

- ❖ Ví dụ: `char str[12];`

## 1. 1. Khai báo và khởi tạo Chuỗi

❖ Ví dụ: `char str[25];`

➤ Ý nghĩa khai báo một mảng kiểu ký tự tên là `str` có 25 phần tử ( như vậy tối đa ta có thể nhập 24 ký tự vì phần tử thứ 25 đã chứa ký tự kết thúc chuỗi `'\0'`).

➤ Lưu ý: Chuỗi ký tự được kết thúc bằng ký tự `'\0'`. Do đó khi khai báo độ dài của chuỗi luôn luôn khai báo dư 1 phần tử để chứa ký tự `'\0'`.

## 1. 1. Khai báo và khởi tạo Chuỗi

### – Cách 2: Dùng con trỏ

```
char *<Tên biến>
```

• *Ví dụ:*     char \*str;

- Trong khai báo này, bộ nhớ sẽ dành 2 byte để lưu trữ địa chỉ của biến con trỏ **str** đang chỉ đến, chưa cung cấp nơi để lưu trữ dữ liệu.



## 1. 1. Khai báo và khởi tạo Chuỗi

### – Cách 2: Dùng con trỏ

- ❖ Trước khi sử dụng phải dùng từ khóa **new** để cấp phát vùng nhớ.

Ví dụ:

```
char *str;
```

```
str = new char[51]; // Cấp phát 51 ký tự
```

## 1. 1. Khai báo và khởi tạo Chuỗi

- ❖ Chuỗi ký tự giống như mảng do đó để khởi tạo một Chuỗi ký tự với những giá trị xác định ta có thể thực hiện tương tự như với mảng.

```
char <Biến>[ ]=<”Hằng Chuỗi ”>
```

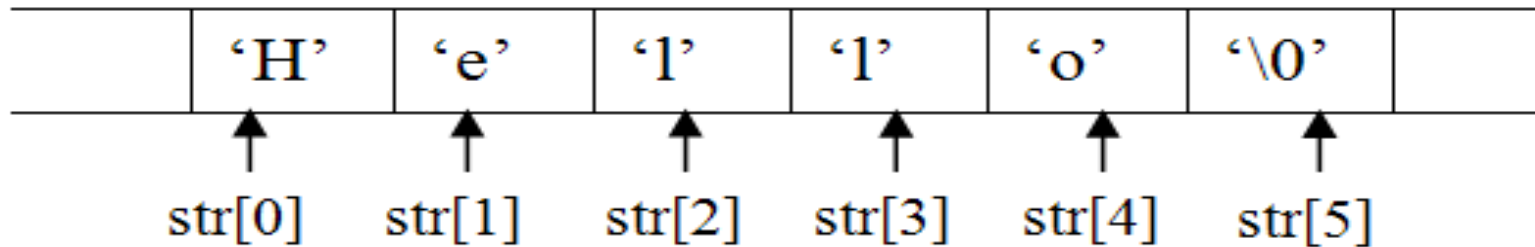
## 1. 1. Khai báo và khởi tạo Chuỗi

❖ Ví dụ:

```
char str[] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

```
char str[] = "Hello";
```

```
char *str = "Hello";
```





❖ Để nhập dữ liệu cho biến Chuỗi, ta dùng hàm **gets()** của thư viện **stdio.h**.

```
char *gets(char *s);
```

❖ Hàm **gets()** đọc các ký tự từ bàn phím vào trong mảng trỏ đến bởi s cho đến khi nhấn Enter. Ký tự **null** sẽ được đặt sau ký tự cuối cùng của Chuỗi nhập vào trong mảng.



❖ Lưu ý: Khi dùng **cin>>** để nhập dữ liệu cho chuỗi, chương trình sẽ tự động ngắt chuỗi khi gặp ký tự khoảng trắng trong chuỗi. Do đó, để chuỗi không bị ngắt khi gặp ký tự khoảng trắng, ta sẽ dùng hàm **gets()**, hoặc **cin.getline()** thay vì hàm **cin** thông thường.

\***cin.getline**(chuỗi, số ký tự tối đa);

\*Ví dụ:

```
char *str;  
str = new char [30];  
cin.getline(str, 30);
```



❖ Để xuất Chuỗi ra màn hình, ta dùng hàm **puts()** của thư viện **stdio.h**.

```
int puts(const char *s);
```

❖ Hoặc ta có thể dùng **cout**  
**cout << s;**



## ❖ Ví dụ:

```
#include <iostream.h>
#include <stdio.h>
int main()
{
    char str[20];
    cout<<"Nhap chuoi:";
    gets(str);
    cout<<"\n Xuat chuoi:";
    puts(str);
    return 0;
}
```



```
#include <iostream.h>
#include <stdio.h>
#define MAX 255
void main()
{
    char str1[MAX];
    char *str2;
    cout<<"Nhap chuoi str1: ";
    gets(str1);
    //cin.getline(str1, MAX);
    str2 = new char[MAX];
    cout<<"Nhap chuoi str2: ";
    cin.getline(str2, MAX);
    cout<<"Chuoi str1: "<<str1<<endl;
    puts(str2);
    //cout<<"Chuoi str2: "<<str2<<endl;
}
```





❖ Để sử dụng các hàm này, ta phải khai báo dòng lệnh sau:

**#include <string.h>**

❖ Sao chép nội dung chuỗi nguồn vào chuỗi đích, nội dung của chuỗi đích sẽ bị xóa.

**strcpy(char \*đích, char \*nguồn);**



❖ Ví dụ: **strcpy(s1, s2):** Sao chép Chuỗi s2 vào s1

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

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

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

```
void main()
```

```
{
```

```
    char str1[20], str2[20];
```

```
    cout<<"nhap chuoi 1:"; gets(str1);
```

```
    strcpy(str2,str1);
```

```
    cout<<"\nXuat chuoi 2:"; puts(str2);
```

```
}
```



❖ Chép n ký tự từ chuỗi nguồn sang chuỗi đích. Nếu chiều dài nguồn < n thì hàm sẽ điền khoảng trắng cho đủ n ký tự vào đích.

**strncpy(char \*đích, char \*nguồn, int n);**

Ví dụ:

```
#include <iostream.h>
#include <stdio.h>
#include <string.h>
void main()
{
    char str1[20], str2[20];
    cout<<"nhap chuoi 1:"; gets(str1);
    strncpy(str2, str1, 5);
    cout<<"\nXuat chuoi 2:"; puts(str2);
}
```



❖ Nối chuỗi s2 vào cuối chuỗi s1

**strcat(s1, s2)**


Ví dụ:

```
#include <iostream.h>
#include <stdio.h>
#include <string.h>
void main()
{
    char str1[20], str2[20];
    cout<<"nhap chuoi 1:"; gets(str1);
    cout<<"\nhap chuoi 2:"; gets(str2);
    strcat(str1,str2);
    cout<<"\nXuat chuoi sau khi noi:";
    puts(str1);
}
```



❖ Nối n ký tự đầu tiên của chuỗi s2 vào chuỗi s1

**strncat(char s1[],char s2[],int n);**

- 
- **strchr(s1, ch)** : Trả về con trỏ đến vị trí xuất hiện đầu tiên của ký tự ch trong Chuỗi s1

*Ví dụ:*

```
void main()
```

```
{
```

```
    char *p, h, str1[20];
```

```
    cout<<"nhap chuoi 1:"; gets(str1);
```

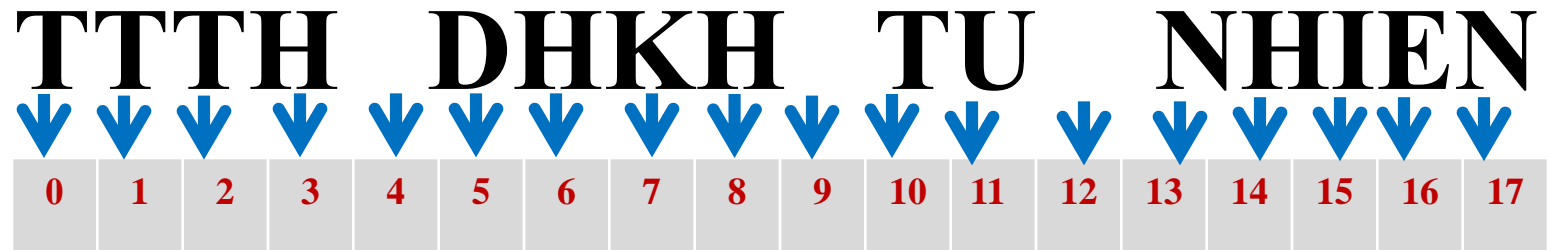
```
    cout<<"Nhap ktu muon tim:"; cin>>h;
```

```
    p= strchr(str1,h);
```

```
    if(p==NULL) cout<<"Khong tim thay ";
```

```
    else cout<<"Tim thay tai vi tri "<<(p-str1);
```

```
}
```





- **strstr(s1, s2):** Trả về con trỏ đến vị trí xuất hiện đầu tiên của Chuỗi s2 trong s1.

Ví dụ: `void main()`

```
{    char *p, str1[20], str2[20];
    cout<<"nhap chuoi 1:"; gets(str1);
    cout<<"nhap chuoi 2:"; gets(str2);
    p= strstr(str1,str2);
    if(p==NULL)
        cout<<"Khong tim thay ";
    else
        cout<<"Tim thay tai vi tri "<<(p-str1);
}
```





❖ Tính độ dài của chuỗi s

**strlen(char \*s);**

```
void main()
```

```
{
```

```
    char *ch = "Lap trinh C";
```

```
    cout<<"Do dai s = "<<strlen(ch);
```

```
}
```

Kết quả

Do dai s = 11



---

\* Nối chuỗi s2 vào chuỗi s1

**strcat(char s1[],char s2[]);**

\*Nối n ký tự đầu tiên của chuỗi s2 vào chuỗi s1

**strncat(char s1[],char s2[],int n);**

\*So sánh 2 chuỗi s1 và s2 theo nguyên tắc thứ tự từ điển. Phân biệt chữ hoa và thường.Trả về :

0 : nếu s1 bằng s2.

=1: nếu s1 lớn hơn s2.

=-1: nếu s1 nhỏ hơn s2.

**strcmp(char s1[],char s2[]);**



```
#include <iostream.h>|
#include <stdio.h>
#include <string.h>
int main()
{
    char *s1="a";
    char *s2="a";
    int ret=strcmp(s1,s2);
    if(ret==0)
    {
        cout<<"2 thang do bang nhau\n";
    }
    else if(ret==1)
    {
        cout<<"thang 1>thang 2\n";
    }
    else if(ret==-1)
    {
        cout<<"thang 1<thang 2\n";
    }
    return 0;
}
```



\*So sánh n ký tự đầu tiên của s1 và s2, giá trị trả về tương tự hàm strcmp()

**strncmp(char s1[],char s2[], int n);**

\*So sánh chuỗi s1 và s2 nhưng không phân biệt hoa thường, giá trị trả về tương tự hàm strcmp()

**stricmp(char s1[],char s2[]);**

\*So sánh n ký tự đầu tiên của s1 và s2 nhưng không phân biệt hoa thường, giá trị trả về tương tự hàm strcmp()

**strnicmp(char s1[],char s2[], int n);**



❖ Chuyển ký tự thường sang ký tự hoa

`toupper( int ch );`

❖ Chuyển ký tự hoa sang ký tự thường

`tolower( int ch );`

❖ Khai báo thư viện: `<ctype.h>`



```
#include <iostream.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
int main()
{
    char s1[]="dai hoc cong nghe dong nai";
    for(int i=0;i<strlen(s1);i++)
    {
        s1[i]=toupper(s1[i]);
    }
    cout<<s1;
    cout<<"\n";
    return 0;
}
```



❖ Mảng các Chuỗi là một mảng ký tự hai chiều. Kích thước của chỉ mục thứ nhất là số Chuỗi và kích thước của chỉ mục thứ hai xác định chiều dài lớn nhất của mỗi Chuỗi.

Ví dụ: `char str[5][80];`

→ Khai báo một mảng của 5 Chuỗi, mỗi Chuỗi có chiều dài tối đa là 79 ký tự.



## ❖ Khai báo và khởi tạo mảng các Chuỗi

```
char arrayList[][length] = {  
                                constantString1,  
                                constantString2,  
                                ...  
                                constantStringN};
```

## ❖ Ví dụ:

```
char listOfPL[][10] = {"Pascal", "C++", "C#"};
```





## ❖ *Vi du:*

```
void main()
{
    char list[5][20];
    for(int i=0; i<5; i++)
    {
        cout<<"name"<<i<<": "; cin>>list[i];
    }
    for(int j=0; j<5; j++)
        cout<<" "<<list[j];
}
```



❖ Ngoài cách dùng mảng ký tự hai chiều để lưu trữ mảng các Chuỗi, ta có thể dùng mảng của các con trỏ. Mỗi con trỏ sẽ chứa địa chỉ của Chuỗi

❖ Ví dụ:

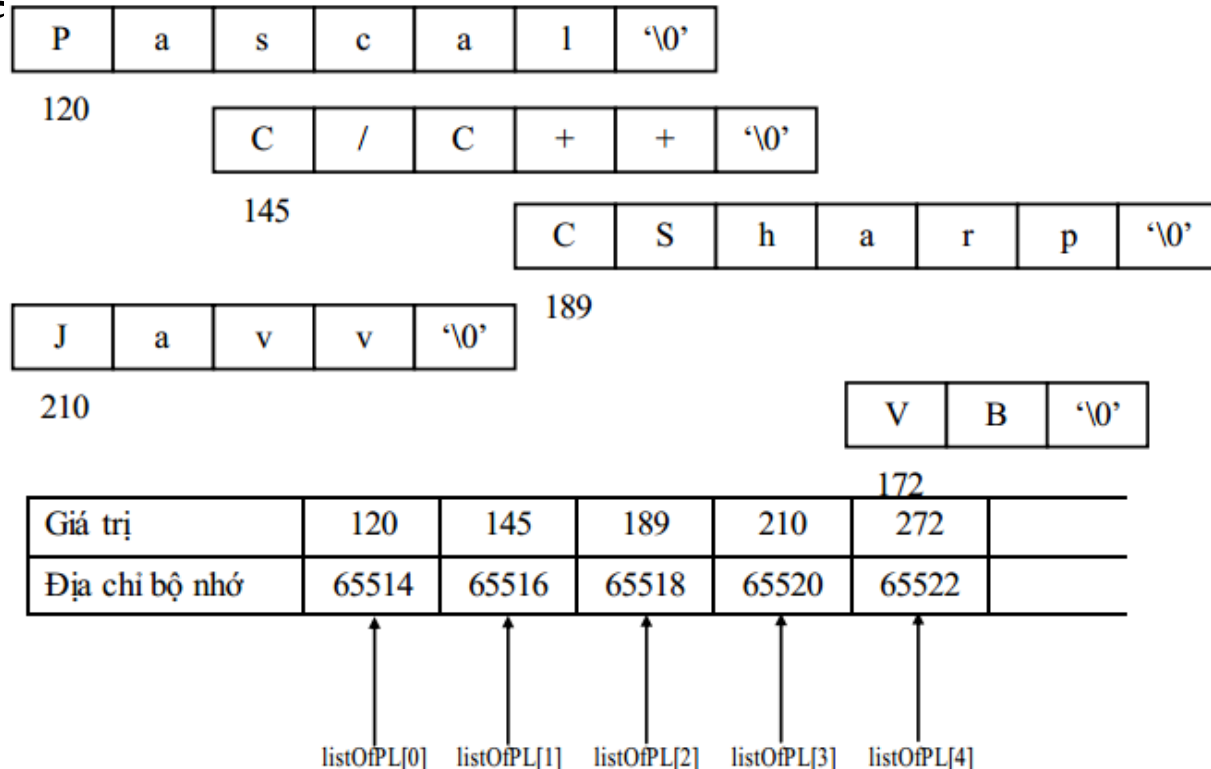
**char \*str[20];**



❖ Cũng ví dụ như phần trên, ta dùng mảng con trỏ

**char \*listOfPL[]** = {“Pascal”, “C/C++”, “CSharp”, “Java”, “VB”};

❖ Mảng con trỏ trên có thể được lưu trữ trong bộ nhớ như sau:





---

```
void main()
```

```
{
```

```
    char *name[5];
```

```
    for(int i=0 ; i<5 ; i++)
```

```
        //name[i] = (char *)malloc(20);
```

```
    name[i] = new char[20];
```

```
    for(int i=0 ; i<5 ; i++)
```

```
    {
```

```
        cout << "Input name " << i+1 << ": ";
```

```
        gets(name[i]);
```

```
    }
```

```
    cout << "List of names: ";
```

```
    for(int i=0 ; i<5 ; i++)
```

```
        cout << name[i] << ", ";
```

```
}
```



## Bài 7:

# Kiểu Dữ Liệu Có Cấu Trúc





**1. Khái niệm kiểu cấu trúc**

**2. Khai báo cấu trúc**

**3. Các thao tác cơ bản**

**4. Mảng cấu trúc và các thao tác**

**5. Enumerations - Typedef**



❖ Ngôn ngữ C/C++ đưa ra 5 cách để tạo nên một kiểu dữ liệu tùy biến (custom data types).

1. **Structure:** Là một nhóm của các biến được định nghĩa dưới một tên. Kiểu này còn gọi là kiểu dữ liệu phức hợp.
2. **Bit-field:** là một biến thể của kiểu **structure** và cho phép dễ dàng truy cập đến từng bit riêng rẽ.
3. **Union:** cho phép cùng một mẫu bộ nhớ được định nghĩa như hai hay nhiều kiểu biến khác nhau.
4. **Enumeration:** là danh sách của các tên hằng nguyên.
5. **Typedef:** định nghĩa một tên khác cho một kiểu dữ liệu đã có.

## 1.2. Khai niệm kiểu cấu trúc



❖ *Ví dụ mở đầu*: Trường đại học XYZ cần viết 1 phần mềm quản lý sinh viên, thông tin cần lưu trữ cho mỗi sinh viên bao gồm: mã sinh viên, họ tên, tuổi, quê quán, lớp học.....

➔ Trong trường hợp này chúng ta cần giải quyết vấn đề : 1 dữ liệu có khả năng chứa nhiều thông tin trong nó.

➔ để giải quyết vấn đề này ta dùng kiểu dữ liệu có **Cấu Trúc**



## 1.2. Khai niệm kiểu cấu trúc



- Một cấu trúc là một tập các biến được tham chiếu thông qua một tên chung. Những biến tạo nên cấu trúc được gọi là các thành viên (*members*).
- Sự khác biệt giữa kiểu cấu trúc và kiểu mảng là: các phần tử của mảng là **cùng kiểu** còn các phần tử của kiểu cấu trúc **có thể có kiểu khác nhau**.

## 2.1. Khai báo kiểu cấu trúc



- ❖ Cấu trúc thực chất là một kiểu dữ liệu do người dùng định nghĩa bằng cách gom nhóm các kiểu dữ liệu cơ bản có sẵn trong C thành một kiểu dữ liệu phức hợp nhiều thành phần.

**struct** **tructureName**

{

**type member1;**

**type member2;**

...

**type memberN;**

.. .

**} varNames;**

- *structureName*: Tên của cấu trúc
- *type*: Kiểu dữ liệu của thành viên tương ứng
- *Member1,..., memberN*: Tên các biến thành viên của cấu trúc
- *varNames*: Tên các biến cấu trúc phân cách nhau bằng dấu phẩy.

## 2. Khai báo kiểu cấu trúc

### Ví dụ:

- Tạo một **struct** sinh viên gồm các thông tin: mã số sinh viên, họ tên, năm sinh, địa chỉ.

**struct sinhvien**

```
{  
    char MaSv[10];  
    char TenSv[30];  
    int Namsinh;  
    char Diachi[50];  
};
```

- Khai báo biến kiểu sinhvien: **sinhvien** sv1, sv2;



```
#include<iostream.h>
typedef struct SinhVien
{
    int ma;
    char ten[50];
    float dtb;
    char lop[50];
};

void main()
{
    int x; //int la primitive data
    SinhVien teo; //Sinhvien la
                //kieu du lieu Doi Tuong
}
```

Từ khóa **typedef** dùng để định nghĩa một kiểu dữ liệu mới

Khi ta khai báo: SinhVien **teo**;

Thì teo chính là 1 đối tượng có kiểu dữ liệu là SinhVien

→ nó tương tự như ta khai báo `int x;`

→ Vậy để lấy các thông tin bên trong của **teo** thì làm như thế nào?

→ Những thông tin mà nằm bên trong 1 cấu trúc (struct) thì được gọi là các thuộc tính của cấu trúc đó

→ Tức là **teo** là 1 đối tượng cụ thể, teo có 4 thuộc tính đó là: ma, ten, dtb, lop

→ để truy xuất tới các thuộc tính ta dùng:

tên\_đối\_tượng.thuộc\_tính



- ❖ Dùng toán tử **dấu chấm** (dot operator) để truy cập các thành viên của một biến cấu trúc.
- ❖ **Cú pháp:**

**varNames.memberName**

- ❖ *Ví dụ:*

```
strcpy(sv1.MaSv, “a001”);  
strcpy(sv1.TenSv, “Nguyen van A”);  
sv2.Namsinh=1977;
```



❖ Dùng lệnh gán để gán nội dung trong 1 biến cấu trúc cho một biến cấu trúc khác có cùng kiểu.

❖ Ví dụ:

```
struct coordXY
{
    int x;
    int y;
} diemA, diemB;
diemA.x = 100; diemA.y = 200;
```



❖ Gán nội dung biến cấu trúc diemA cho biến diemB:

$\text{diemB} = \text{diemA};$

❖ Hoặc có thể sao chép từng thành viên:

$\text{diemB.x} = \text{diemA.x};$

$\text{diemB.y} = \text{diemA.y};$



```
#include<iostream.h>
typedef struct SinhVien
{
    int ma;
    char ten[50];
    float dtb;
    char lop[50];
};
void main()
{
    int x; //int la primitive data
    SinhVien teo;//Sinhvien la
                //kieu du lieu Doi Tuong
    teo.ma=1;
    //teo.ten="nguyen thi teo"==>SAI, ko cho kieu nay
    strcpy(teo.ten,"nguyen thi teo");
    teo.dtb=10;
    strcpy(teo.lop,"tcth40a");
}
```

❖ Cách nhập dữ liệu có  
kiểu chuỗi từ bàn phím:  
dùng **gets**





❖ Viết chương trình nhập vào tọa độ hai điểm trong mặt phẳng và tính tổng hai tọa độ này

\*Bước 1: Khai báo kiểu dữ liệu có cấu trúc biểu diễn thông tin tọa độ của một điểm trong mặt phẳng gồm 2 thành phần: hoành độ và tung độ

```
typedef struct ttDIEM
{
    int x;
    int y;
};
ttDIEM DIEM;
```



## \*Bước 2: Cài đặt các hàm:

- Nhập vào tọa độ điểm

void Nhap (DIEM &d);

- Xuất tọa độ điểm

void Xuat (DIEM d);

- Tính tổng hai tọa độ

DIEM Tong (DIEM d1,DIEM d2);



```
void Nhap (DIEM &d)
{
    cout<<"Hoanh do : ";
    cin>>d.x;
    cout<<"Tung do : ";
    cin>>d.y;
}
```



```
void Xuat (DIEM d)
{
    cout<<“\nToa do diem : (“ <<d.x<< “,”<<d.y<<”)”;
}
```

```
DIEM Tong (DIEM d1,DIEM d2)
{
    DIEM temp;
    temp.x = d1.x + d2.x ;
    temp.y = d1.y + d2.y ;
    return Temp;
}
```



```
void main ()
{
    DIEM A , B, AB; //khai bao 3 diem A, B, AB;
    cout<<"Nhap toa do diem thu 1: "<<endl;
        Nhap ( A );
        Xuat ( A );
    cout<<"Nhap toa do diem thu 2: "<<endl;
        Nhap ( B );
        Xuat ( B );
    cout<<"\n Tong toa do cua hai diem vua nhap la : ";
        AB = Tong ( A, B);
        Xuat ( AB );
}
```



- Mảng cấu trúc là một mảng mà mỗi phần tử là một biến kiểu cấu trúc.
- Để khai báo một mảng các cấu trúc, trước hết phải khai báo cấu trúc, sau đó khai báo một mảng của cấu trúc đó.

Ví dụ: **struct** ds{

**char** hoten[25];

**float** toan,ly,hoa;

};

ds bangdiem[50]; //mảng 50 phần tử kiểu ds



❖ Để truy cập đến từng thành viên của từng phần tử của mảng, ta dùng chỉ mục của phần tử và toán tử thành viên (.).

Ví dụ:

```
for(int i=0; i<n; i++)  
{  
    cout <<"Ho va ten: "; gets(bangdiem[i].hoten)  
    cout <<"Diem toan: "; cin >> bangdiem[i].toan;  
    cout <<"Diem ly:    "; cin >> bangdiem[i].ly;  
    cout <<"Diem hoa    "; cin >> bangdiem[i].hoa;  
}
```



## **a. Truyền thành viên của biến cấu trúc vào hàm**

- Có 2 cách truyền thành viên của biến cấu trúc vào hàm:
  - Truyền tham trị
  - Truyền tham biến





## *Ví dụ: Truyền tham trị*

```
struct diem
```

```
{
```

```
    int x;
```

```
    int y;
```

```
};
```

```
double khcach(int x1, int y1, int x2, int y2)
```

```
{
```

```
    double kc;
```

```
    kc=sqrt(pow((x2-x1),2)+pow((y2-y1),2));
```

```
    return kc;
```

```
}
```



## *Ví dụ: Truyền tham trị*

```
void main()
{
    diem a,b;    double kcach;
    cout<<"\ntoa do diem a:";
    cout<<"\nx1=";cin>>a.x;
    cout<<"\ny1=";cin>>a.y;
    cout<<"\ntoa do diem b:";
    cout<<"\nx2=";cin>>b.x;
    cout<<"\ny2=";cin>>b.y;
    cout<<"\nKhoang cach giua a diem:";
        kcach=khcach(a.x, a.y,b.x, b.y);
    cout<<kcach;
}
```



## *Ví dụ: truyền tham chiếu*

- ❖ Để truyền địa chỉ của thành viên của cấu trúc vào hàm ta dùng toán tử & đặt trước tên biến cấu trúc

```
void doitoado(int &x, int& y, int a, int b)
```

```
{      x=x-a; y=y-b; }
```

```
void main()
```

```
{  diem a,b;
```

```
    doitoado(a.x, a.y, 10, 10);
```

```
    cout<<"\nx="<<a.x;
```

```
    cout<<"\ny="<<a.y;
```

```
}
```



## b. Truyền toàn bộ biến cấu trúc đến hàm

Khi một cấu trúc được dùng như một đối số của một hàm, toàn bộ cấu trúc được truyền vào tham số hình thức. Có hai cách truyền

– *Truyền tham trị.*

```
double khcach(diem a,diem b)
{
    double kc;
    kc=sqrt(pow((b.x-a.x),2)+pow((b.y-a.y),2));
    return kc;
}
```



– *Truyền tham chiếu*

```
void doitoado(diem &a,int n, int m)
```

```
{
```

```
    a.x=a.x-n;    a.y=a.y-m;
```

```
}
```

```
void main()
```

```
{
```

```
    diem a,b;
```

```
    doitoado(a, 10, 10);
```

```
    cout<< "x="<<a.x<<" y="<<a.y;
```

```
}
```



❖ Một biến con trỏ có thể trỏ đến một biến kiểu cấu trúc.

❖ *Cú pháp khai báo một con trỏ cấu trúc*

**structureName \*structurePointers;**

❖ Ví dụ:

```
struct diem
```

```
{
```

```
    int x;
```

```
    int y;
```

```
};
```

diem \*p; // p là con trỏ cấu trúc.



❖ Để tham chiếu đến thành viên của một cấu trúc được trỏ đến bởi một con trỏ, ta dùng toán tử  $\rightarrow$  (toán tử tham chiếu gồm một dấu trừ và một dấu lớn hơn).

❖ Ví dụ:

$P \rightarrow x = 100;$

$P \rightarrow y = 150;$



```
void main()
{
    struct diem
    {
        int x;
        int y;
    };
    diem *p, a;
    p=&a;
    p->x=100;
    p->y=120;
    cout<<a.x;
    cout<<a.y;
}
```





## ❖ Lưu ý:

- Để truy cập đến thành viên của một cấu trúc:
  - Nếu dùng *biến cấu trúc* thì dùng toán tử *chấm* (dot operator).
  - Nếu dùng *biến con trỏ* thì dùng toán tử *->* (arrow operator).
- Truyền tham số là con trỏ cấu trúc thì mặc định là truyền *tham chiếu*



- Một **enum** là một tập của các tên hằng nguyên xác định tất cả các giá trị hợp lệ mà một biến của kiểu đó có thể có.
- Cú pháp:

**enum** **enumName** {enumList} enumVars;

- *enum*: từ khóa để khai báo enum
- *enumName*: Tên của enum
- *enumList*: Danh sách các tên hằng nguyên phân cách nhau bởi dấu phẩy
- *enumVars*: Tên các biến kiểu enum.



- *Vi du:*

```
enum color {red, orange, yellow, green, blue, indigo};  
color c1 = indigo;  
if( c1 == indigo )  
{  
    cout << "c1 is indigo" << endl;  
}
```



- Mỗi một tên trong danh sách enum tượng trưng cho một giá trị nguyên. Giá trị của tên thứ nhất trong enum là 0, kế tiếp là 1, ...
- Ta có thể gán giá trị khác cho mỗi tên hằng nguyên



- Từ khóa **typedef** dùng để định nghĩa một tên mới cho một kiểu dữ liệu đã có.
- Dạng tổng quát của dùng typedef là  
**typedef existingType newType;**
  - *existingType*: là kiểu dữ liệu nào đã tồn tại
  - *newType*: tên mới của kiểu dữ liệu



- Ví dụ: Tạo một tên mới cho kiểu dữ liệu int

```
typedef int int2bytes;
```

```
typedef long int4bytes;
```

Sau khi các lệnh trên thực hiện thì lệnh

```
int n1;
```

```
long n2;
```

tương đương

```
int2bytes n1;
```

```
Int4bytes n2;
```

# Viết chương trình quản lý sinh viên



```
#include<iostream.h>
#include <stdio.h>
#define SISOTOIDA 100
typedef struct SinhVien
{
    int ma;
    char ten[50];
    float dtb;
    char lop[50];
};

void nhapthongtin(SinhVien &sv);
void xuatthongtin(SinhVien sv);
void nhapdssv(SinhVien ds[SISOTOIDA],int siso);
void xuatdssv(SinhVien ds[SISOTOIDA],int siso);
void main()
{
    SinhVien lopTcth40a[SISOTOIDA];
    int sisothucte;
    cout<<"Nhap si so : ";
    cin>>sisothucte;
    if(sisothucte>SISOTOIDA)
    {
        cout<<"Dong qua sao ma day";
        exit(0);
    }
    cout<<"Moi ban nhap danh sach:\n";
    nhapdssv(lopTcth40a,sisothucte);
    cout<<"Danh sach lop sau khi nhap:\n";
    xuatdssv(lopTcth40a,sisothucte);
}
```

```
void nhapthongtin(SinhVien &sv)
{
    cout<<"Nhap ma: ";
    cin>>sv.ma;
    cout<<"Nhap ten:";
    gets(sv.ten);
    cout<<"Dtb:";
    cin>>sv.dtb;
    cout<<"Nhap lop:";
    gets(sv.lop);
}

void xuatthongtin(SinhVien sv)
{
    cout<<"ma: "<<sv.ma<<"\n";
    cout<<"ten:"<<sv.ten<<"\n";
    cout<<"Dtb:"<<sv.dtb<<"\n";
    cout<<"lop:"<<sv.lop<<"\n";
}

void nhapdssv(SinhVien ds[SISOTOIDA],int siso)
{
    for(int i=0;i<siso;i++)
    {
        cout<<"Nhap sv thu "<<(i+1) <<": \n";
        nhapthongtin(ds[i]);
    }
}

void xuatdssv(SinhVien ds[SISOTOIDA],int siso)
{
    for(int i=0;i<siso;i++)
    {
        cout<<"\n-----\n";
        xuatthongtin(ds[i]);
        cout<<"-----\n";
    }
}
```

