

# Thành Nguyen

## Kỹ thuật lập trình

---

### Chương 2: Các yếu tố cơ bản của C và C++



$$y = A*x + B*u;$$
$$x = C*x + d*u;$$

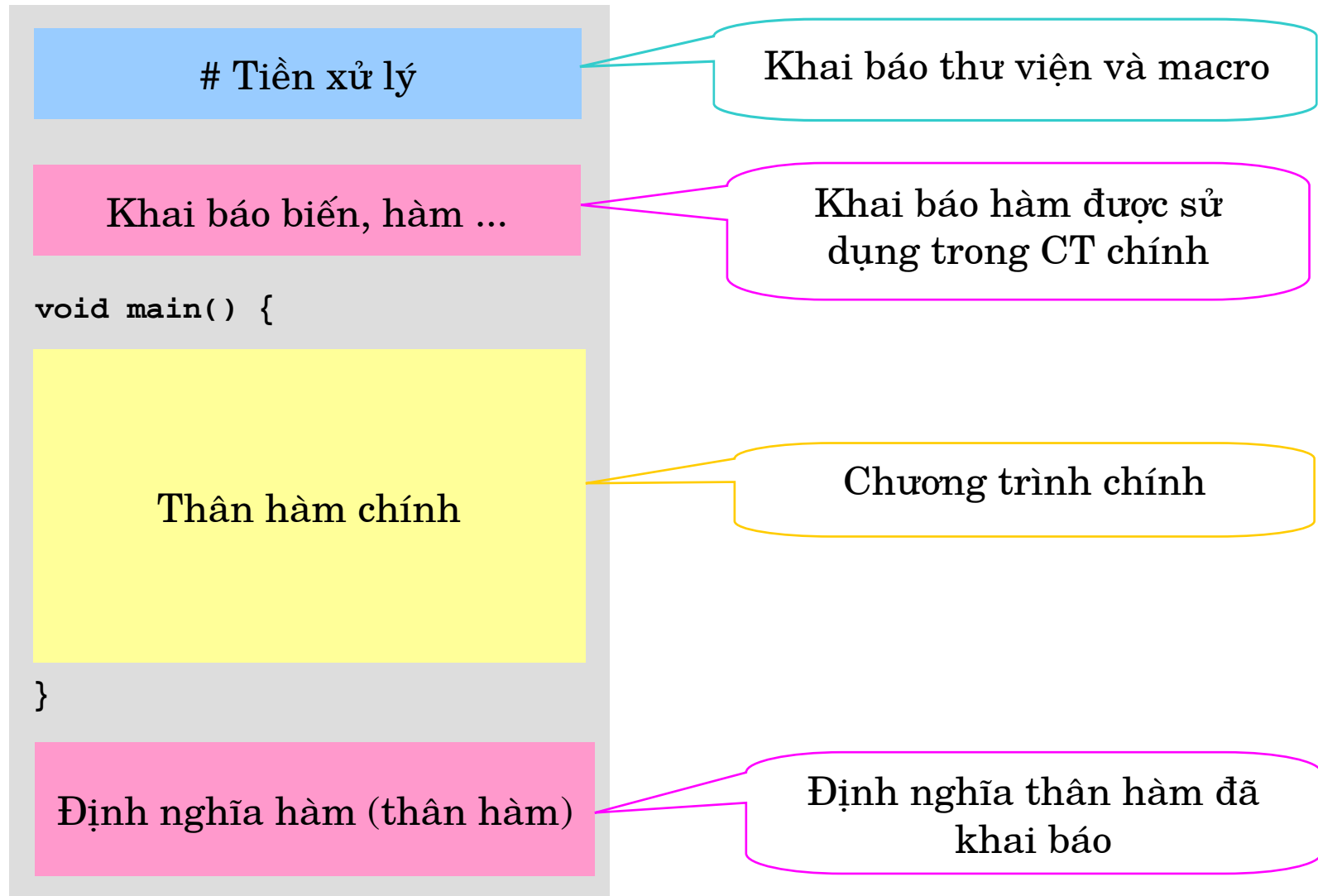
# Nội dung chương 2

- 2.1 Tổ chức chương trình C/C++
- 2.2 Biến và các kiểu dữ liệu cơ bản
- 2.3 Các kiểu dữ liệu dẫn xuất trực tiếp
- 2.4 Định nghĩa kiểu dữ liệu mới
- 2.5 Điều khiển chương trình: phân nhánh
- 2.6 Điều khiển chương trình: vòng lặp
- 2.7 Một số lệnh điều khiển chương trình khác

# 2.1 Tổ chức chương trình C/C++

- Cấu trúc và các phần tử cơ bản của một chương trình viết trên C/C++
- Quy trình tạo ra một chương trình chạy được:
  - Vấn đề tạo dự án
  - Quy tắc soạn thảo mã nguồn
  - Biên dịch từng phần và sửa các loại lỗi biên dịch
  - Liên kết và sử dụng thư viện, sửa lỗi liên kết
  - Chạy thử và gỡ rối (Debug)
- Sơ lược về tổ chức bộ nhớ

# 2.1 Tổ chức chương trình C/C++



# Chương trình tính giai thừa: Phiên bản C

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

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

```
void main() {
```

```
    char c = 'N';
    int N = 1;
    int kq;
    do {
        printf("\nEnter a number > 0:");    /* writing on the screen */
        scanf("%d",&N);                      /* reading from keyboard to N */
        kq = factorial(N);                   /* calling function with argument N */
        printf("\nFactorial of %d is %d", N, kq); /*write result on screen */
        printf("\nPress 'Y' to continue or any other key to stop");
        c = getch();                         /* reading a character from keyboard*/
    } while (c=='y' || c=='Y');              /* checking loop condition */
```

```
    }
    int factorial(int n) {
        int kq = 1;
        while (n > 1)
            kq *= n--;
        return kq;
    }
```

# Chương trình tính giai thừa: Phiên bản C++

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

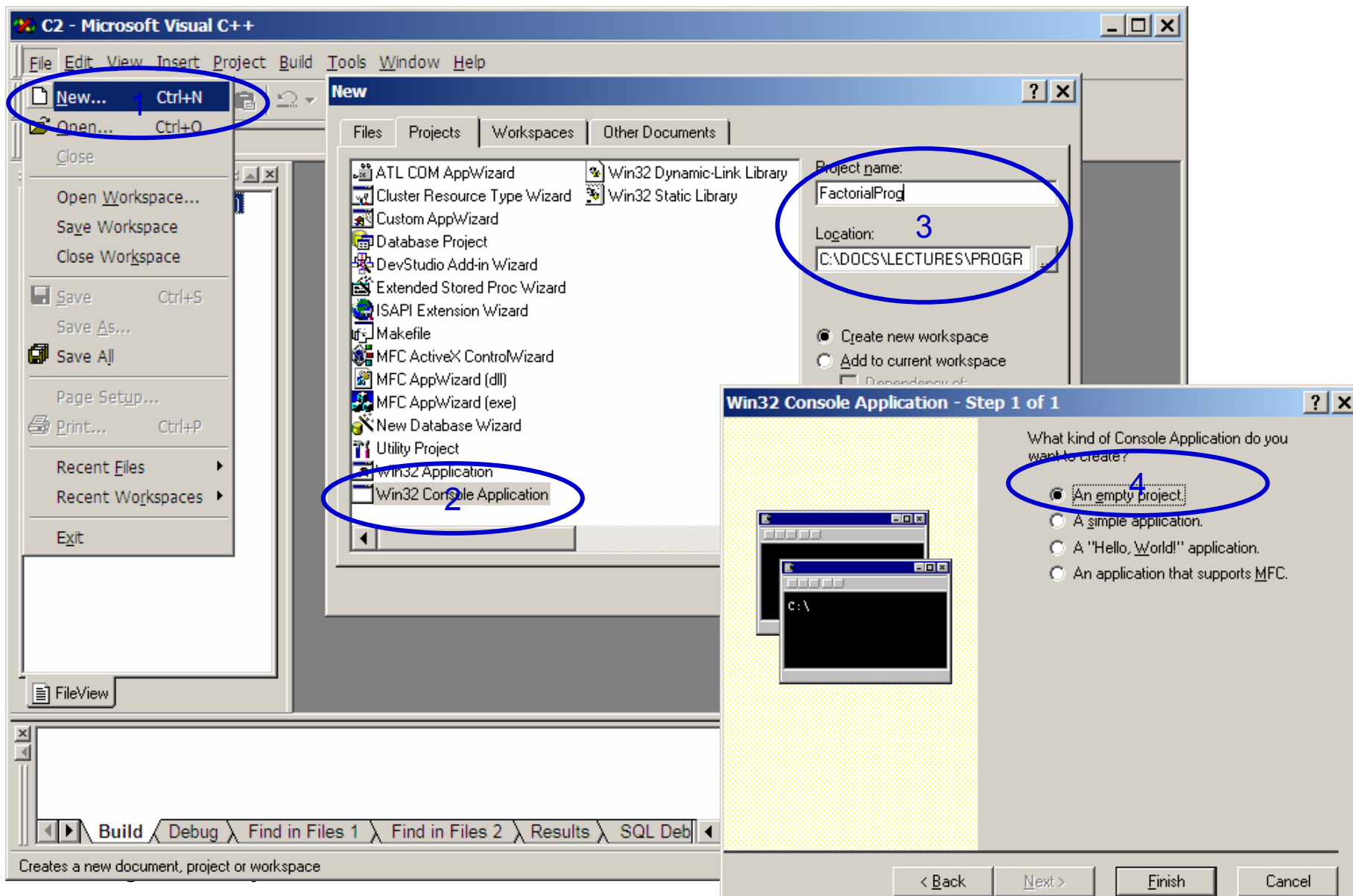
```
int factorial(int);
```

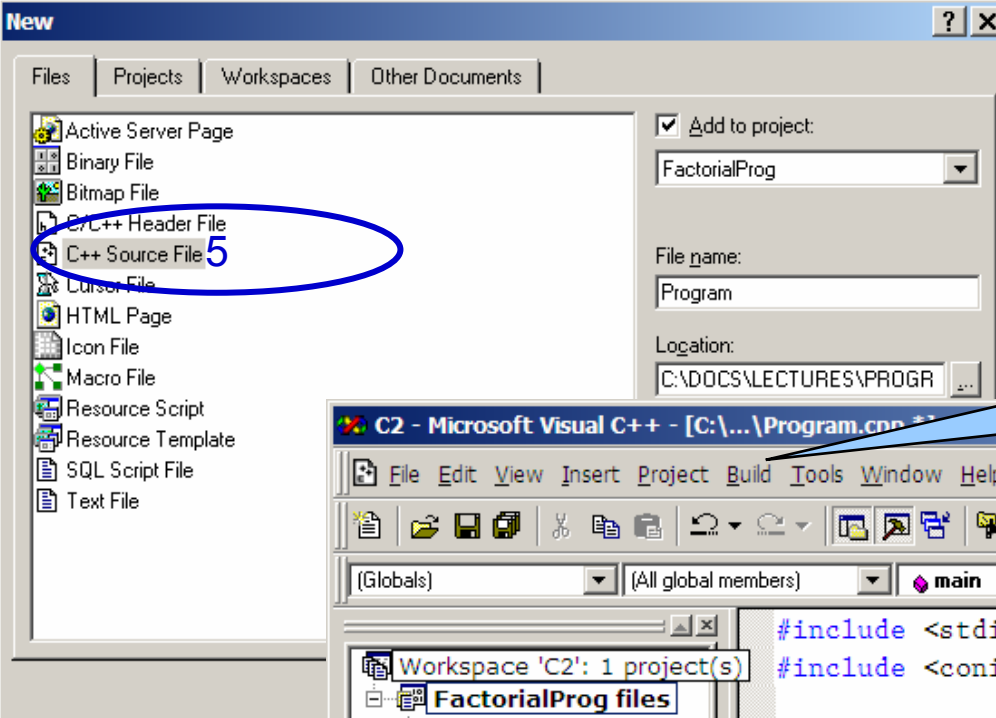
```
void main() {
```

```
    char c = 'N';
    int N = 1;
    do {
        cout << "\nEnter a number > 0:" // writing on the screen
        cin >> N;                        // reading from keyboard to N
        int kq = factorial(N);           // calling function with argument
        cout << "\nFactorial of " << N << " is " << kq
        cout << "\nPress 'Y' to continue or any other key to stop";
        c = getch();                    // reading a character from keyboard
    } while (c == 'y' || c == 'Y');      // checking loop condition
}
```

```
int factorial(int n) {
    int kq = 1;
    while (n > 1)
        kq *= n--;
    return kq;
}
```

# Tạo dự án

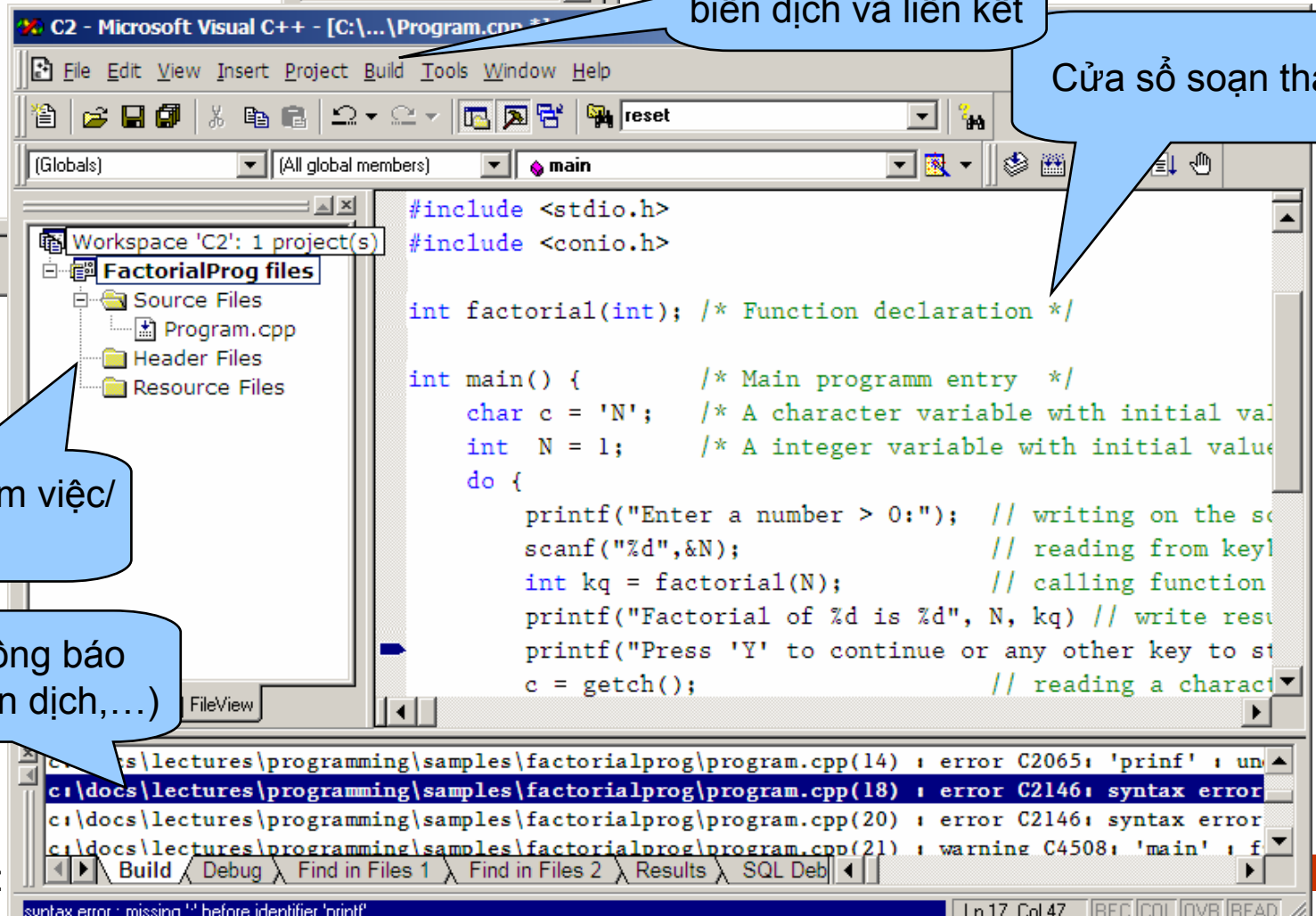




# Bổ sung file mã nguồn và soạn thảo

Các công cụ  
biên dịch và liên kết

Cửa sổ soạn thảo



Cửa sổ bàn làm việc/  
dự án

Cửa sổ thông báo  
kết quả (biên dịch,...)



# Qui tắc soạn thảo mã nguồn

1. Tên biến, tên hàm, tên kiểu mới:
  - Tránh sử dụng các từ khóa và tên kiểu cơ sở
  - Các ký tự dùng được: 'A'..'Z', 'a'..'z', '0'..'9', '\_'
  - Phân biệt giữa chữ hoa và chữ thường: **n** khác **N**
  - Ngắn nhưng đủ khả năng phân biệt, gợi nhớ để nhận biết
  - Sử dụng tiếng Anh hoặc tiếng Việt không dấu (kể cả dòng chú thích)
2. Sau mỗi câu lệnh có chấm phẩy;
3. Đoạn { ... } được coi là nhóm lệnh, không có dấu chấm phẩy sau đó, trừ trường hợp khai báo kiểu
4. Cấu trúc mã nguồn theo kiểu phân cấp => dễ đọc
5. Bổ sung chú thích đầy đủ, hợp lý (`/* ...*/` hoặc `//`)
6. Chia một file lớn thành nhiều file nhỏ

# Các từ khóa trong C

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

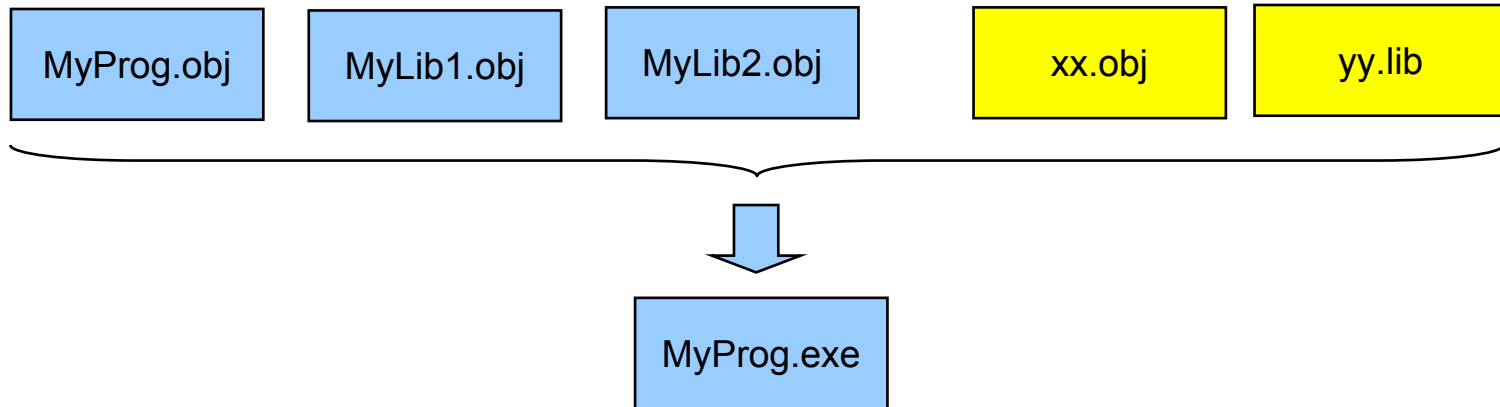
# Từ khóa trong C++

<b>asm</b>	<b>auto</b>	bool	break
case	<b>catch</b>	char	<b>class</b>
const	<b>const_cast</b>	continue	default
<b>delete</b>	else	extern	do
enum	<b>false</b>	double	<b>explicit</b>
float	<b>dynamic_cast</b>	<b>export</b> for	
<b>friend</b>	goto	if	<b>inline</b>
int	long	<b>mutable</b>	<b>namespace</b>
<b>new</b>	<b>operator</b>	<b>private</b>	<b>protected</b>
<b>public</b>	register	<b>reinterpret_cast</b>	return
short	signed	sizeof	static
<b>static_cast</b>	struct	switch	<b>template</b>
<b>this</b>	<b>throw</b>	<b>true</b>	<b>try</b>
typedef	<b>typeid</b>	<b>typename</b>	union
unsigned	<b>using</b>	<b>virtual</b>	void
volatile	<b>wchar_t</b>	while	

# Biên dịch (compile)

- Biên dịch từng file nguồn riêng biệt (\*.c: C compiler, \*.cpp: C++ compiler), kết quả => \*.obj
- Trong Visual C++: Gọi **Compile** (**Ctrl + F7**) để biên dịch riêng rẽ hoặc **Build** (**F7**) để kết hợp biên dịch và liên kết cho toàn bộ dự án
- Các kiểu lỗi biên dịch (compile error):
  - Lỗi cú pháp: Sử dụng tên sai qui định hoặc chưa khai báo, thiếu dấu chấm phẩy ;, dấu đóng }
  - Lỗi kiểu: Các số hạng trong biểu thức không tương thích kiểu, gọi hàm với tham số sai kiểu
  - ...
- Các kiểu cảnh báo biên dịch (warning):
  - Tự động chuyển đổi kiểu làm mất chính xác
  - Hàm khai báo có kiểu trả về nhưng không trả về
  - Sử dụng dấu = trong trường hợp nghi vấn là so sánh ==
  - ...

# Liên kết (link)

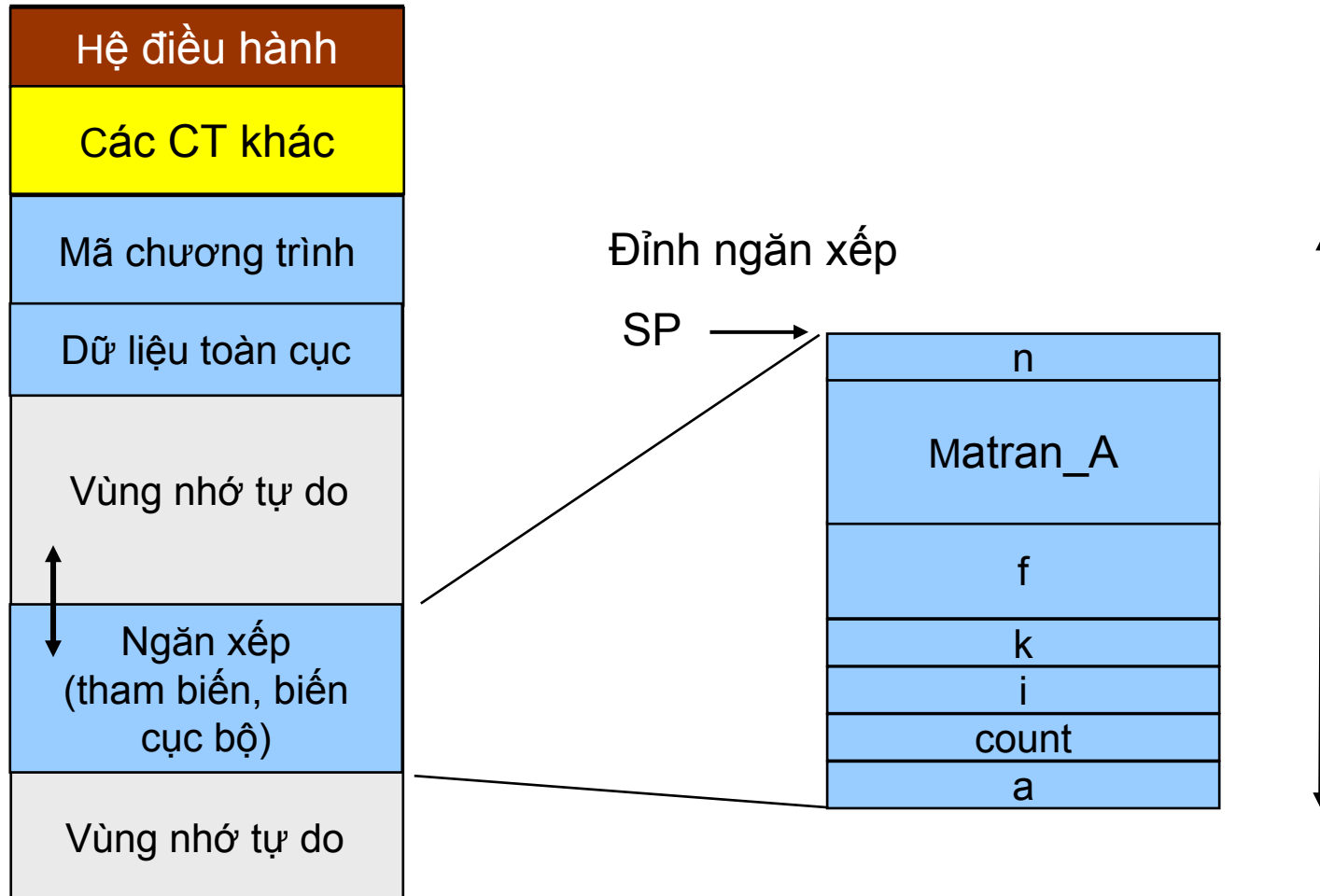


- Liên kết là quá trình ghép nhiều file đích (\*.obj, \*.lib) để tạo ra chương trình chạy cuối cùng \*.exe
- Trong Visual C++: Gọi **Build (F7)**
- Lỗi liên kết có thể là do:
  - Sử dụng hàm nhưng không có định nghĩa hàm
  - Biến hoặc hàm được định nghĩa nhiều lần
  - ...

# Chạy thử và gỡ rối (debug)

- Chạy thử trong Visual C++: **Execute** hoặc **Ctrl+F5**
- Tìm lỗi:
  - Lỗi khi chạy là lỗi thuộc về phương pháp, tư duy, thuật toán, không phải về cú pháp
  - Lỗi khi chạy bình thường không được báo
  - Lỗi khi chạy rất khó phát hiện, vì thế trong đa số trường hợp cần tiến hành **debug**.
- Chạy Debug trong Visual C++:
  - Chạy tới chỗ đặt cursor: **Ctrl+F10**
  - Chạy từng dòng lệnh: **F10**
  - Chạy vào trong hàm: **F11**
  - Chạy tiếp bình thường: **F5**
  - Xem kết quả dưới cửa sổ Output hoặc gọi QuickWatch

# Tổ chức bộ nhớ



## 2.2 Biến và dữ liệu

- Biểu thức = dữ liệu + phép toán + ...
- Biểu diễn dữ liệu: Thông qua **biến** hoặc **hằng số**, kèm theo **kiểu**
- Nội dung trong phần này:
  - Các kiểu dữ liệu cơ bản
  - Các phép toán áp dụng
  - Tương thích và chuyển đổi kiểu
  - Khai báo biến, phân loại biến



## 2.2.1 Các kiểu dữ liệu cơ bản của C/C++

Kiểu	Kích cỡ thông dụng (tính bằng bit)	Phạm vi tối thiểu
<b>char</b>	8	-127 to 127
<b>signed char</b>	8	-127 .. 127
<b>unsigned char</b>	8	0 .. 255
<b>int</b>	16/32	-32767 .. 32767
<b>signed int</b>	16/32	-nt-
<b>unsigned int</b>	16/32	0 .. 65535
<b>short</b>	16	-32767 .. 32767
<b>signed short</b>	16	nt
<b>unsigned short</b>	16	0 .. 65535
<b>long</b>	32	-2147483647..2147483647
<b>signed long</b>	32	- nt-
<b>unsigned long</b>	32	0 .. 4294967295
<b>float</b>	32	Độ chính xác 6 chữ số
<b>double</b>	64	Độ chính xác 15 chữ số
<b>long double</b>	80	Độ chính xác 17 chữ số
<b>bool</b> (C++)	-	-
<b>wchar_t</b> (C++)	16	-32767 .. 32767

# Các phép toán cơ bản

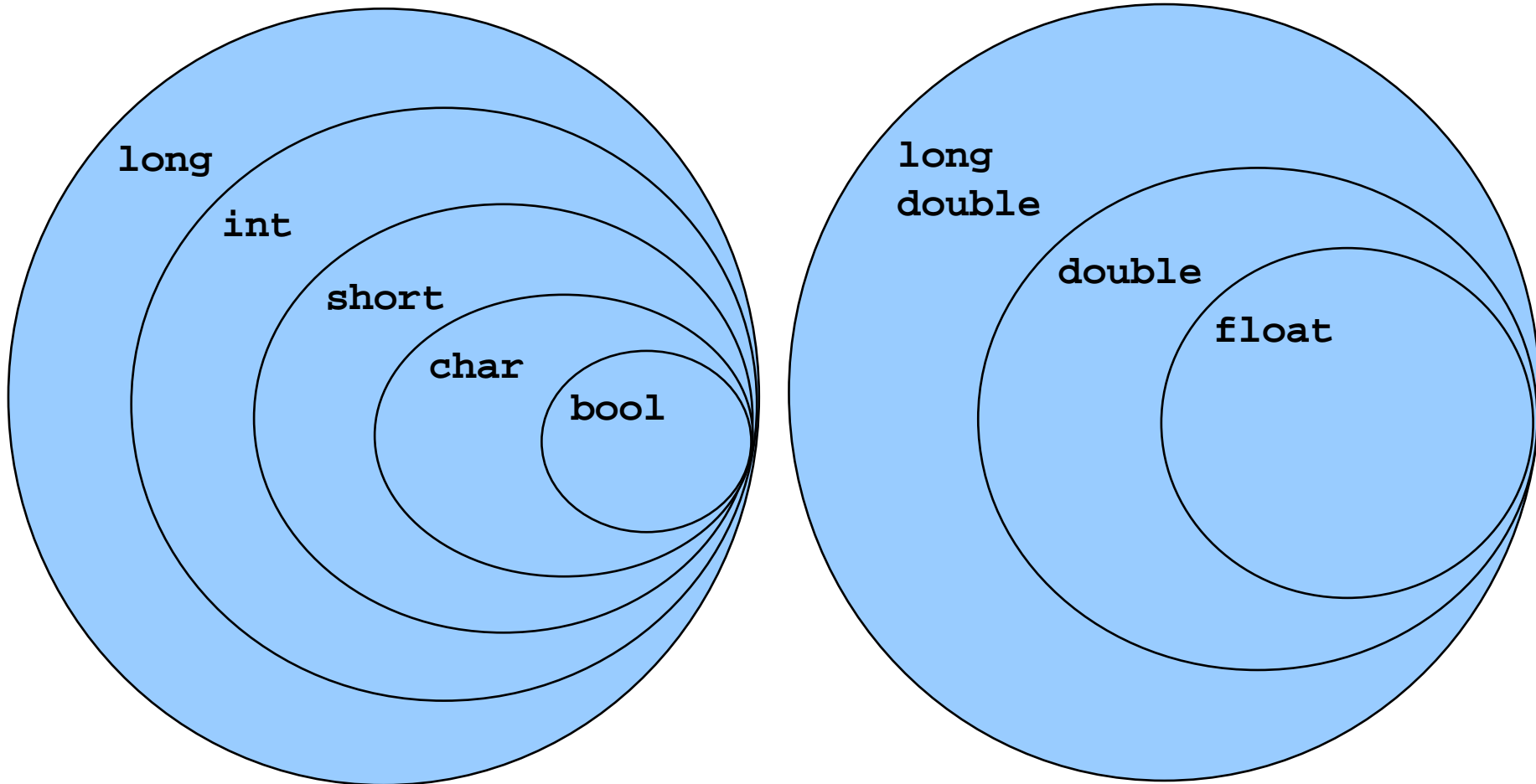
Phép toán	Ký hiệu	Kiểu nguyên	Kiểu số thực	Kiểu bool
Gán	=	X	X	X
Số học	+, -, *, /, +=, -=, *=, /=	X	X	x
	%, %=	X		x
	++, --	X		x
So sánh	>, <, >=, <=, ==, !=	X	X	X
Logic	&&,   , !	X	X	X
Logic bit	&,  , ^, ~ &=,  =, ^=	X		x
Dịch bit	<<, >>, <<=, >>=	X		x
Lựa chọn	? :	X	X	X
Lũy thừa?	<b>Không có!</b>			

# Tương thích và chuyển đổi kiểu

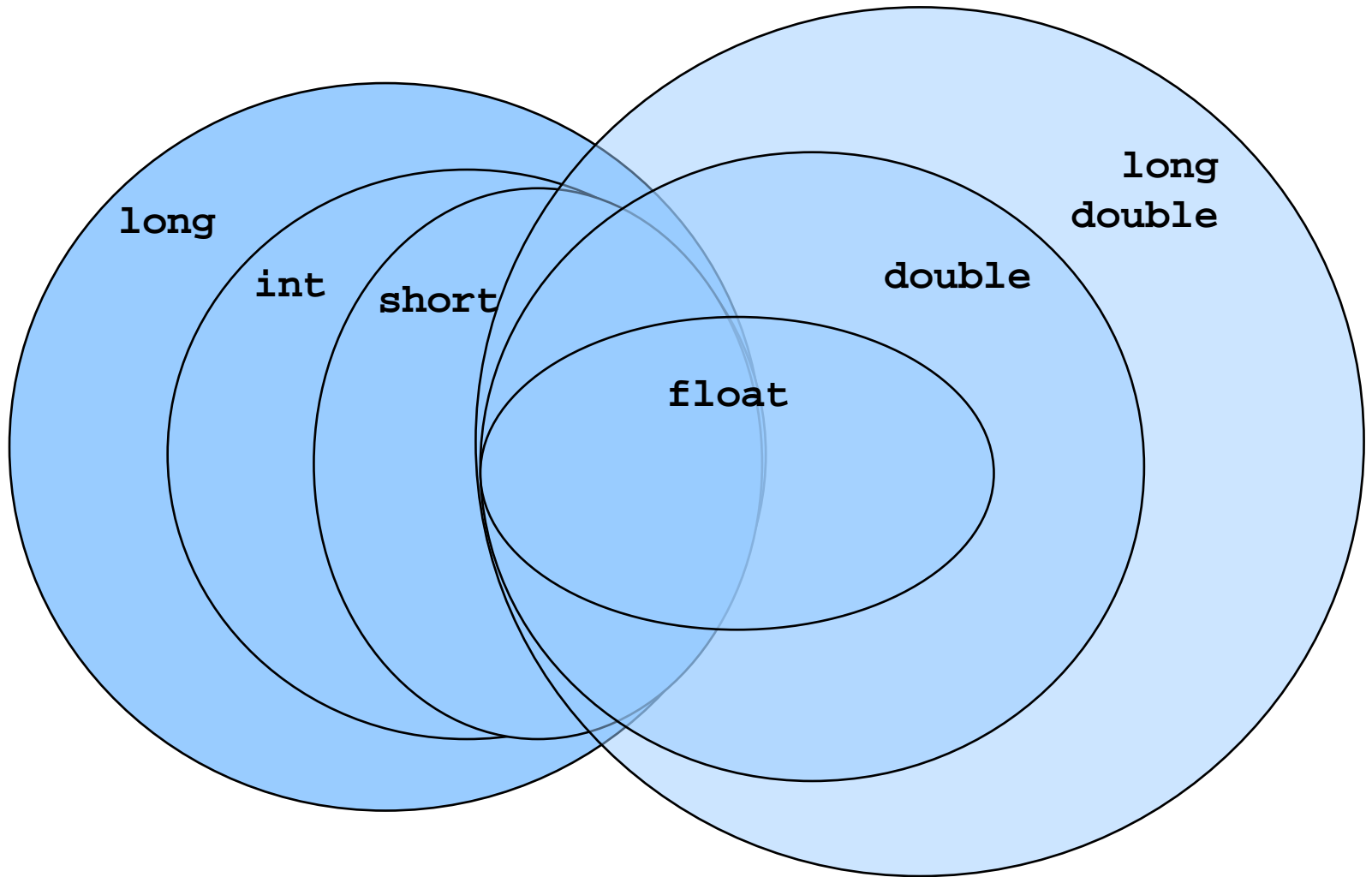
- Tương thích kiểu => Tự động chuyển đổi kiểu
  - Giữa các kiểu số nguyên với nhau (lưu ý phạm vi giá trị)
  - Giữa các kiểu số thực với nhau (lưu ý độ chính xác)
  - Giữa các kiểu số nguyên và số thực (lưu ý phạm vi giá trị và độ chính xác)
  - Kiểu bool sang số nguyên, số thực: true => 1, false => 0
  - Số nguyên, số thực sang kiểu bool:  $\neq 0$  => true, 0 => false
- Nếu có lỗi hoặc cảnh báo => khắc phục bằng cách ép chuyển đổi kiểu:
  - VD:

```
i = int(2.2) % 2;  
j = (int)2.2 + 2; // C++
```

# Nhìn nhận về chuyển đổi kiểu



# Nhìn nhận về chuyển đổi kiểu



## 2.2.2 Khai báo biến

<code>char</code>	<code>c = 'N';</code>	←	Khai báo và khởi tạo giá trị
<code>bool</code>	<code>b = true;</code>	←	
<code>int</code>	<code>kq;</code>	←	Chỉ khai báo, giá trị bất định
<code>double</code>	<code>d;</code>	←	
<code>long</code>	<code>count, i=0;</code>	←	Khai báo kết hợp, chỉ <code>i=0</code>
<code>unsigned</code>	<code>vhexa=0x00fa;</code>	←	Đặt giá trị đầu hexa
<code>unsigned</code>	<code>vocal=082;</code>	←	Đặt giá trị đầu octal -> 66 chứ không phải 82

- C: Toàn bộ biến phải khai báo ngay đầu thân hàm
- C++: Có thể khai báo tại chỗ nào cần, trước khi sử dụng
- Phân loại biến:
  - Biến toàn cục: Khai báo ngoài hàm, lưu giữ trong vùng nhớ dữ liệu chương trình
  - Biến cục bộ: Khai báo trong thân hàm, lưu giữ trong ngăn xếp
  - Tham biến: Khai báo trên danh sách tham số của hàm, lưu giữ trong ngăn xếp

# Ví dụ khai báo các loại biến

Biến toàn cục

Biến cục bộ

Hai biến cục bộ  
cùng tên ở hai phạm  
vi khác nhau,  
không liên quan gì  
đến nhau!

```
int N = 1;
void main() {
    char c = 'N';
    do {
        printf("\nEnter a number > 0:");
        scanf("%d",&N);
        int kq = factorial(N); // C++ only!
        ...
    } while (c == 'y' || c == 'Y')
}

int factorial(int n) {
    int kq = 1;
    while (n > 1)
        kq *= n--;
    return kq;
}
```

Tham biến

# Đặc tính lưu giữ

- Biến **extern**: Khai báo sử dụng biến toàn cục đã được định nghĩa và gán giá trị trong một tập tin khác

```
/* file1.c */
int x, y;
char ch;
void main()
{
    /* ... */
}
void func1(void)
{
    x = 123;
}
```

```
/* file2.c */
extern int x, y;
extern char ch;
void func22()
{
    x = y / 10;
}
void func23()
{
    y = 10;
}
```

- Biến **static**: được lưu trữ trong bộ nhớ dữ liệu CT
  - Biến static cục bộ: hạn chế truy nhập từ bên ngoài hàm
  - Biến static toàn cục: hạn chế truy nhập từ file khác



## 2.2.3 Hằng số (trực kiện)

Kiểu	Ví dụ
int	1 123 21000 -234 0x0A 081
long int	35000L -341 -234L 0x0AL 081L
unsigned int	10000U 987u 40000u
float	123.23F 4.34e-3f .1f
double	123.23 1.0 -0.9876324 .1e-10
long double	1001.2L
char	'A' 'B' ' ' 'a' '\n' '\t' '\b'
bool	true false
wchar_t	L'A' L'B'

## 2.3 Các kiểu dữ liệu dẫn xuất trực tiếp

- Kiểu liệt kê
- Kiểu hằng
- Kiểu con trỏ
- Kiểu mảng
- Kiểu tham chiếu (C++)

## 2.3.1 Kiểu liệt kê (enum)

- Mục đích sử dụng:
  - Định nghĩa một kiểu là tập các hằng số nguyên kí hiệu
  - Sử dụng thuận tiện bằng tên => **hằng số nguyên**
- Ví dụ

```
enum Color {Red, Green, Blue};
enum WeekDay {
    Mon = 2,
    Tue, Wed, Thu, Fri, Sat,
    Sun = 1 };
enum {
    DI_MOTOR1_STARTED = 0x01,
    DI_MOTOR1_RUNNING = 0x02,
    DI_MOTOR2_STARTED = 0x04,
    DI_MOTOR2_RUNNING = 0x08,
    DI_PUMP1_STARTED = 0x10,
    DI_PUMP1_RUNNING = 0x20,
    DI_OVERLOADED = 0x40,
    DI_VALVE1_OPENED = 0x80
};
```

# Sử dụng kiểu liệt kê

/\* C version \*/

```
void main() {  
    enum Color c = Red;           /* c = 0 */  
    enum WeekDay d = Tue;         /* d = 3 */  
    int i=c, j=d;                 /* j=0, i=3 */  
    enum Color c2 = i+1;          /* c2 = 1 */  
    int di1 = 0x01;               /* OK, but... */  
    int di2 = DI_MOTOR1_STARTED; /* this is better */  
    ++c;                          /* c = 1 */  
}
```

C:  
Như một kiểu số  
nguyên 8 bit

// C++ version \*/

```
void main() {  
    enum Color c = Red;           // c = Red  
    WeekDay d = Tue;              // OK, d = Tue  
    int i=c, j=d;                 // i=0, j=3  
    Color c2 = i+1;               // Error!  
    Color c3 = Color(i+1);        // OK, c3 = Green  
    int di1 = 0x01;               // OK, but...  
    int di2 = DI_MOTOR1_STARTED; // this is better  
    ++c;                          // Error!  
}
```

C++  
Không còn như  
một kiểu số  
nguyên!

## 2.3.2 Kiểu hằng (const)

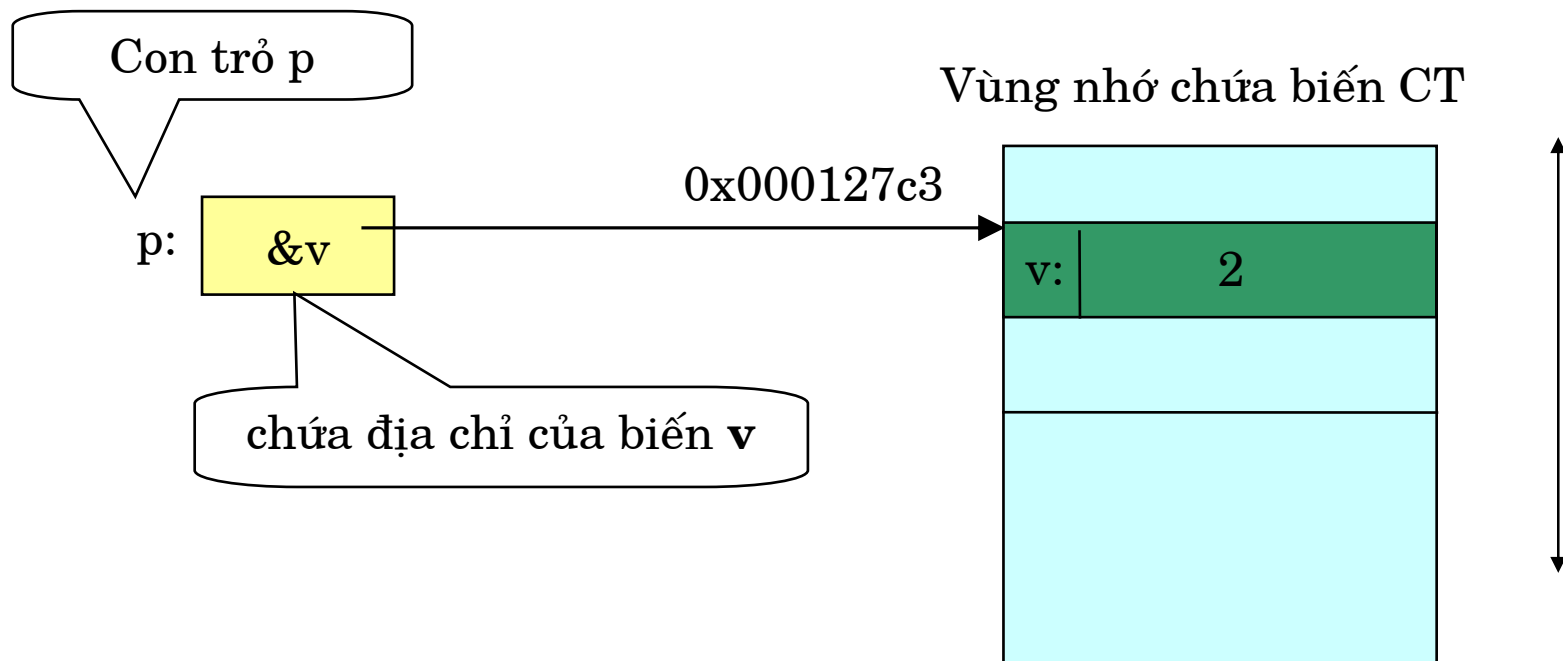
- Cho phép khai báo như biến số, nhưng được gán giá trị cố định bằng một hằng số và không thể được thay đổi => khai báo hằng số

```
void main() {  
    const double pi = 3.1412;    // initializing is OK!  
    const int ci = 1;            // initializing is OK!  
    ci = 2;                      // error!  
    ci = 1;                      // error, too!  
    int i = ci;                  // const int is a subset of int  
    const Color cc = Red;  
    cc = Green;                  // error  
    const double d;              // potential error  
}
```

## 2.3.3 Kiểu con trỏ

- Con trỏ thực chất là một biến chứa địa chỉ của một đối tượng có thể là một biến hoặc một hàm.

```
int v = 2;  
int* p = &v; // p holds the address of v
```



## 2.3.3 Kiểu con trỏ

```
int i = 1;
int* p = &i; // p holds the address of i
*p = 2;      // i = 2
int j;
p = &j;      // now p holds the address of j
*p = 3;      // j = 3, i remains 2
```

Variables

Context: main()

Name	Value
&i	0x0012ff7c
i	1

Auto Locals this

Variables

Context:

Name	Value
i	2
p	0x0012ff7c

Auto Locals this

Variables

Context: main()

Name	Value
j	3
i	2
p	0x0012ff74

Auto Locals this

Variables

Context: main()

Name	Value
&i	0x0012ff7c
i	1
p	0x0012ff7c
*p	1

Auto Locals this

Variables

Context: main()

Name	Value
j	-858993460
&j	0x0012ff74
p	0x0012ff74
*p	-858993460

Auto Locals this

# Ví dụ sử dụng kiểu con trỏ

```
void main() {
    int i = 0;
    int* p = &i; // p refers to the address of i
    int j = *p; // j = 0
    *p = 2; // now i = 2
    p = &j; // now p contains the address of j
    *p = 3; // now j = 3, i remains 2
    double d = i; // OK, int is compatible to double
    p = &d; // error, int* isn't compatible to double*
    p = (*int)&d; // no compile error, but dangerous,
                // meaningles type conversion!

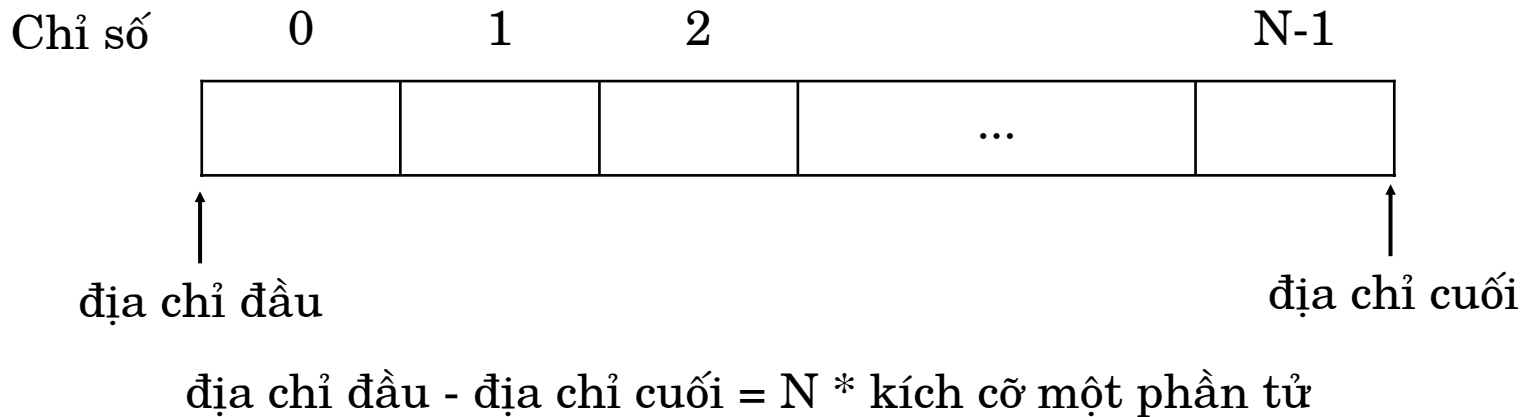
    double* pd=0; // p contains the address 0
    *pd = 0; // no compile error, but fatal error
    pd = &d; // OK
    double* pd2; // p refers to an uncertain address
    *pd2 = 0; // fatal error
    pd2 = &d; // OK, pd and pd2 refer to the same addr.
}
```



# Tóm tắt sơ bộ về con trỏ

- Con trỏ là một biến chứa địa chỉ byte đầu của một biến dữ liệu, được sử dụng để truy cập gián tiếp dữ liệu đó
- Sau khi khai báo mà không khởi tạo, mặc định con trỏ mang một địa chỉ bất định
- Địa chỉ con trỏ mang có thể thay đổi được => con trỏ có thể mỗi lúc đại diện cho một biến dữ liệu khác
- Toán tử lấy địa chỉ của một biến (&) trả về con trỏ vào kiểu của biến => thường gán cho biến con trỏ
- Toán tử truy nhập nội dung (\*) áp dụng cho con trỏ trả về biến mà con trỏ mang địa chỉ => có thể đọc hoặc thay đổi giá trị của biến đó
- Không bao giờ sử dụng toán tử truy nhập nội dung, nếu con trỏ chưa mang một địa chỉ ô nhớ mà chương trình có quyền kiểm soát

## 2.3.4 Kiểu mảng



- Cấu trúc dữ liệu với:
  - Số lượng các phần tử cố định
  - Các phần tử có cùng kiểu
  - Các phần tử được sắp xếp kế tiếp trong bộ nhớ
  - Có thể truy nhập từng phần tử một cách tự do theo chỉ số hoặc theo địa chỉ

# Khái báo mảng

- Số phần tử của mảng phải là hằng số nguyên (trong C phải là một trực kiện, trong C++ có thể là kiểu **const** ...)
- Khái báo không khởi tạo:

```
int          a[3];  
enum         {index = 5};  
double       b[index];  
const int    N = 2;  
char         c[N]; // C++ only
```

- Khái báo với số phần tử và khởi tạo giá trị các phần tử

```
int          d[3] = {1, 2, 3};  
double       e[5] = {1, 2, 3};  
char         f[4] = {0};
```

The image shows two screenshots of a debugger's 'Variables' window, likely from Visual Studio, illustrating the state of memory during program execution.

**Top Screenshot:**

- Context: main()
- Variables:

  - N: 2
  - c: 0x0012ff44 "iiii"
    - [0]: -52 'i'
    - [1]: -52 'i'
  - Index: 5
  - b: 0x0012ff4c

**Bottom Screenshot:**

- Context: main()
- Variables:

  - f: 0x0012ff0c ""
    - [0]: 0 ''
    - [1]: 0 ''
    - [2]: 0 ''
    - [3]: 0 ''
  - e: 0x0012ff10
    - [0]: 1.0000000000000000
    - [1]: 2.0000000000000000
    - [2]: 3.0000000000000000
    - [3]: 0.0000000000000000
    - [4]: 0.0000000000000000
  - d: 0x0012ff38
    - [0]: 1
    - [1]: 2
    - [2]: 3

# Khai báo mảng (tiếp)

- Khai báo và khởi tạo giá trị các phần tử, số phần tử được tự động xác định

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

```
double b[] = {1, 2, 3};
```

```
double c[] = {0};
```

```
char s[] = {'a'};
```

- Khai báo mảng nhiều chiều

```
double M[2][3];
```

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

```
short T[2][2] = {1, 2, 3, 4, 5, 6};
```

Name	Value
M	0x0012ff50
[0]	0x0012ff50
[0]	-9.2559631349318e+061
[1]	-9.2559631349318e+061
[2]	-9.2559631349318e+061
[1]	0x0012ff68
[0]	-9.2559631349318e+061
[1]	-9.2559631349318e+061
[2]	-9.2559631349318e+061
X	0x0012ff38
[0]	0x0012ff38
[0]	1
[1]	2
[1]	0x0012ff40
[0]	3
[1]	4
[2]	0x0012ff48
[0]	5
[1]	6
T	0x0012ff30
[0]	0x0012ff30
[0]	1
[1]	2

# Ví dụ sử dụng kiểu mảng

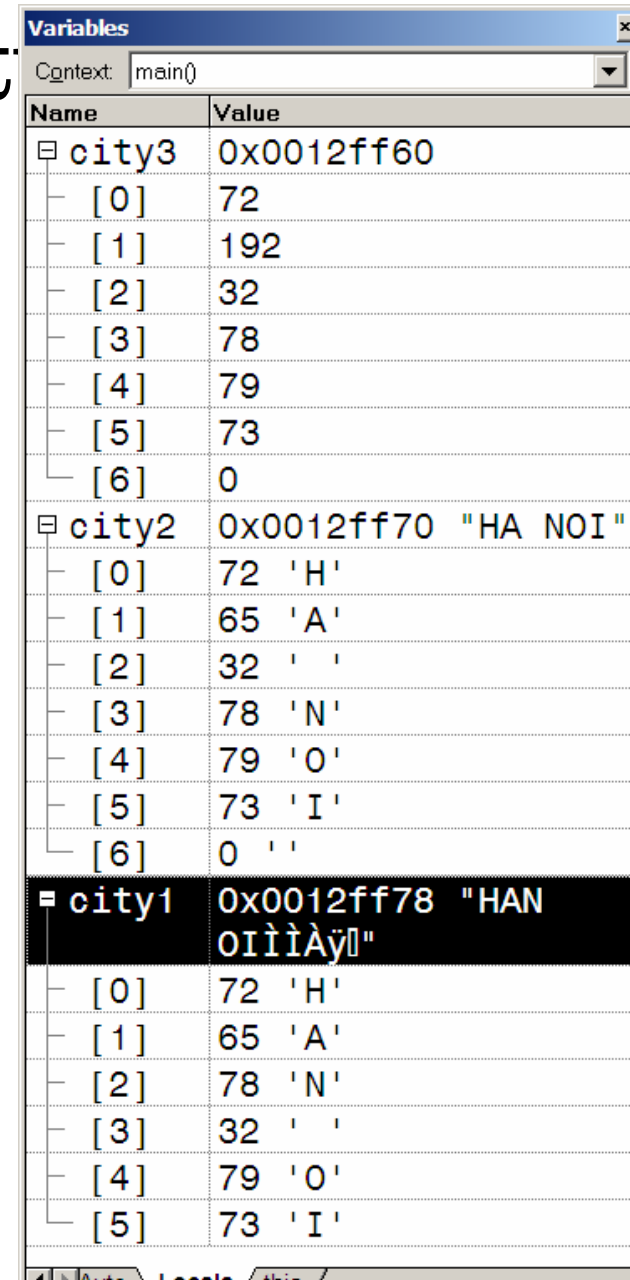
```
void main() {  
    int a[5];           // a has 5 elements with uncertain values  
    int b[5] = {1,3,5,7,9}; // 5 elements with initial values  
    double c[];         // error, unspecified size  
    double x = 1.0, y = 2.0;  
    double d[] = {x,y,3.0}; // 3 elements with initial values  
    short n = 10;  
    double v[n];        // error, array size must be a constant!  
    const int m=10;     // C++ OK  
    double v2[m];       // C++ OK  
    a[0] = 1;  
    int i = 1;  
    a[i] = 2;  
    a[5] = 6;           // no compile error, but fatal error  
    int k = a[5];       // no compile error, but fatal error  
    a = {1,2,3,4,5};    // error  
    a = b;              // error, cannot assign array  
    int M[2][3];  
    M[0][1] = 0;  
    M[0][2] = 1;  
}
```

# Mảng đặc biệt: Chuỗi ký tự

- Trong C/C++, chuỗi ký tự không phải là kiểu cơ bản, mà thực chất là một mảng
- Phân biệt chuỗi ký tự thường và **chuỗi ký tự kết 0**

```
char city1[] = {'H', 'A', 'N', ' ', 'O', 'I'};  
char city2[] = "HA NOI";  
wchar_t city3[] = L"HÀ NOI";  
city2[] = "HANOI"; // error
```

- Đa số các hàm trong thư viện C làm việc với chuỗi ký tự kết 0
- Với C++, chuỗi ký tự được định nghĩa bằng lớp **string** trong thư viện chuẩn, không sử dụng byte kết 0

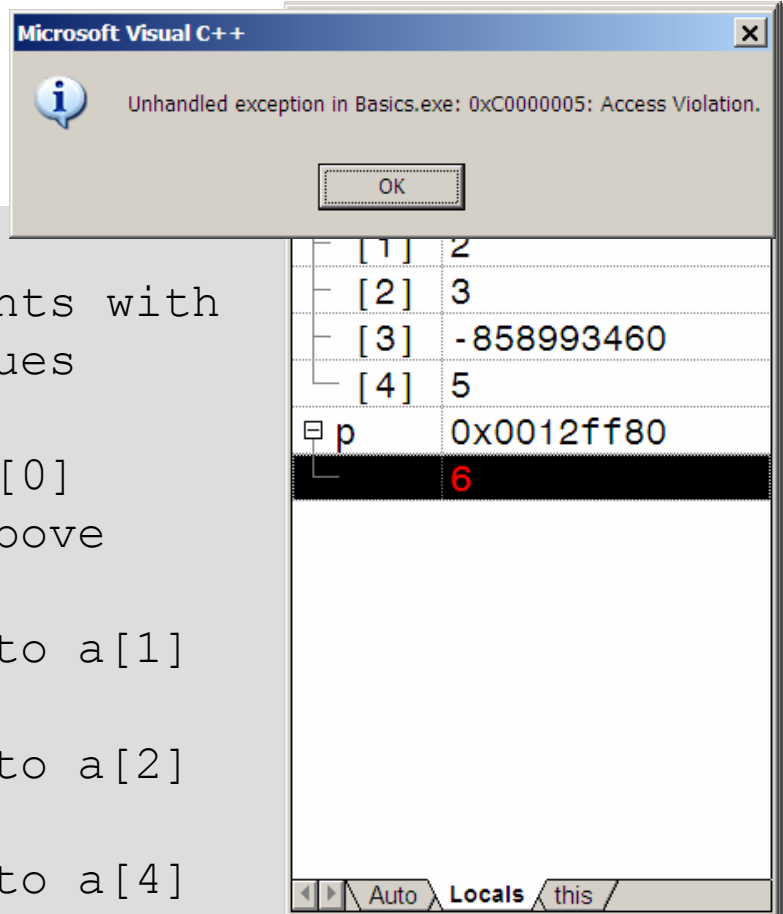


Variables	
Context: main()	
Name	Value
city3	0x0012ff60
[0]	72
[1]	192
[2]	32
[3]	78
[4]	79
[5]	73
[6]	0
city2	0x0012ff70 "HA NOI"
[0]	72 'H'
[1]	65 'A'
[2]	32 ' '
[3]	78 'N'
[4]	79 'O'
[5]	73 'I'
[6]	0 ''
city1	0x0012ff78 "HAN OIììÀÿ"
[0]	72 'H'
[1]	65 'A'
[2]	78 'N'
[3]	32 ' '
[4]	79 'O'
[5]	73 'I'

# Mảng và con trỏ

```
void main() {
    int a[5];    // a has 5 elements with
                // uncertain values

    int* p;
    p = a;       // p refers to a[0]
    p = &a[0];   // the same as above
    *p = 1;      // a[0]=1
    ++p;         // now p points to a[1]
    *p = 2;      // a[1]=2
    p++;         // now p points to a[2]
    *p = 3;      // a[2]=3
    p += 2;      // now p points to a[4]
    *p = 5;      // a[4] = 5
    ++p;         // OK, no problem until we dereference it
    *p = 6;      // Now is a BIG BIG problem!
    a = p;       // error, a is like a constant pointer
}
```



# Mảng và con trỏ (tiếp)

```
void main() {  
    int a[5];    // a has 5 elements with  
                // uncertain values  
    int* p = a;  // p points to a[0]  
    p[0] = 1;    // a[0]=1  
    p[1] = 2;    // a[1]=2  
    p+= 2;       // now p points to a[2]  
    p[0] = 3;    // a[2]=3  
    p[1] = 4;    // a[3]=4  
    p[3] = 6;    // a[5]=6, Now is a BIG BIG problem!  
}
```



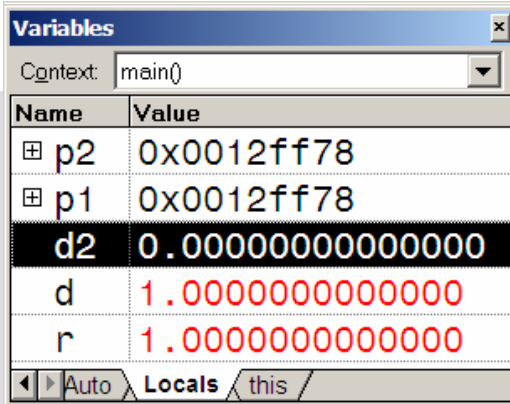
# Tóm lược về mảng

- Mảng là một tập hợp các dữ liệu cùng kiểu, sắp xếp liên kề trong bộ nhớ => các phần tử của mảng
- Có thể truy cập các phần tử mảng với biến mảng kèm theo chỉ số hoặc với biến con trỏ (theo địa chỉ của từng phần tử)
- Số phần tử của mảng là cố định (khi khai báo phải là hằng số), không bao giờ thay đổi được
- Biến mảng (tĩnh) thực chất là một con trỏ hằng, mang địa chỉ của phần tử đầu tiên
- Có thể đặt giá trị đầu cho các phần tử của mảng qua danh sách khởi tạo, không bao giờ gán được mảng cho nhau. Nếu cần sao chép hai mảng thì phải sử dụng hàm
- Không bao giờ được phép truy nhập với chỉ số nằm ngoài phạm vi, nếu  $N$  là số phần tử thì phạm vi cho phép là từ  $0..N-1$
- Con trỏ không bao giờ là một mảng, nó chỉ có thể mang địa chỉ của một mảng và sử dụng để quản lý mảng (dù là động hay tĩnh)

## 2.3.5 Kiểu tham chiếu (C++)

- Một biến tham chiếu là một biến đại diện trực tiếp cho một biến khác (thay cho con trỏ)
- Ý nghĩa sử dụng chủ yếu về sau trong truyền tham số cho hàm

```
void main() {  
    double d = 2.0;  
    double& r = d;    // r represents d  
    double *p1 = &d, *p2 = &r;  
    r = 1.0;          // OK, d = 1.0  
    double& r2;        // error, r has to be assigned to a var.  
    double& r3 = 0;    // error, too  
    double d2 = 0;  
    r = d2;           // r = 0, d=0  
    r = 1.0;          // r = d = 1, d2 =0  
}
```



Variables	
Context: main()	
Name	Value
p2	0x0012ff78
p1	0x0012ff78
d2	0.0000000000000000
d	1.0000000000000000
r	1.0000000000000000

## 2.3.6 Typedef

- Từ khóa **typedef** tạo ra một tên mới cho một kiểu có sẵn, không định nghĩa một kiểu mới
- Ý nghĩa: đưa tên mới dễ nhớ, phù hợp với ứng dụng cụ thể, dễ thay đổi về sau

```
typedef float REAL;  
typedef int AnalogValue;  
typedef int Vector[10];  
typedef AnalogValue AnalogModule[8];  
typedef int* IPointer;  
AnalogValue av1 = 4500;  
Vector x = {1,2,3,4,5,6,7,8,9,10};  
AnalogModule am1 = {0};  
IPointer p = &av1;
```

## 2.4 Định nghĩa kiểu dữ liệu mới

- Cấu trúc (**struct**): Tập hợp những dữ liệu hỗn hợp, truy nhập theo tên (biến thành viên). Thông dụng nhất trong C, ý nghĩa được mở rộng trong C++
- Hợp nhất (**union**): Một tên kiểu chung cho nhiều dữ liệu khác nhau (chiếm cùng chỗ trong bộ nhớ). Ít thông dụng trong cả C và C++
- Lớp (**class**): Chỉ có trong C++, mở rộng **struct** cũ thêm những hàm thành viên.

## 2.4.1 Cấu trúc (struct)

- Định nghĩa cấu trúc (bên trong hoặc ngoài các hàm)

```
struct Time
{
    int hour; // gio
    int minute; // phut
    int second; // giay
};
```

Tên kiểu mới  
(không trùng lặp)

```
struct Date {
    int day, month, year;
};
```

Các biến thành viên,  
khai báo độc lập  
hoặc chung kiểu

```
struct Student {
    char name[32];
    struct Date birthday;
    int id_number;
};
```

Các biến thành viên  
có thể cùng kiểu  
hoặc khác kiểu

C++

# Khai báo biến cấu trúc

```
void main() {  
    Time clasTime = {6,45,0};  
    Time lunchTime = {12};  
    Date myBirthday, yourBirthday = {30,4,1975};  
    Student I = {"Nguyen Van A", {2,9,1975}};  
    //...
```

Variables		Variables		Variables		Variables	
Cgntext: main()		Cgntext: main()		Cgntext: main()		Cgntext: main()	
Name	Value	Name	Value	Name	Value	Name	Value
clasTime	{...}	clasTime	{...}	clasTime	{...}	clasTime	{...}
hour	6	I	{...}	I	{...}	I	{...}
minute	45	myBirthday	{...}	myBirthday	{...}	name	0x0012ff20 "Nguyen Van A"
second	0	yourBirthda	{...}	day	-858993	birthday	{...}
I	{...}	lunchTime	{...}	month	-858993	day	2
myBirthday	{...}	hour	12	year	-858993	month	9
yourBirthda	{...}	minute	0	yourBirthda	{...}	year	1975
lunchTime	{...}	second	0	day	30	id_number	0
				month	4	myBirthday	{...}
				year	1975		

# Sử dụng biến cấu trúc

```
/...  
void main() {  
    Time classTime = {6,45,0};  
    Time lunchTime = {12};  
    Date myBirthDay, yourBirthDay = {30,4,1975};  
    Student I = {"Nguyen Van A", {2,9,1975}};  
    lunchTime.minute = 15;  
    lunchTime.hour = classTime.hour + 6;  
    Student U = I;    // in C++ also possible: Student U(I);  
    U.name[11] = 'B'; // "Nguyen Van B"  
    U.id_number++;    // 1  
    U.birthDay.day = 30;    // 30-9-1975  
    U.birthDay.month = 4;    // 30-4-1975  
    U.birthDay = yourBirthDay; // structs can be assigned  
}
```

# Phản ví dụ: khai báo và sử dụng cấu trúc

```
struct Time {  
    int hour = 0;        // error, initialization not allowed  
    int minute,          // error, use semicolon (;) instead  
    int second           // error, missing semicolon (;)  
} // error, missing semicolon (;)  
//...  
void main() {  
    Date d;  
    d = {11,9,2001}; // error, {...} is an initialization  
                  // list, not a structure  
    Date.hour = 0;    // error, Date is a type, not a var.  
    struct Date2 { int day, month, year; };  
    Date2 d2 = d; // error, Date is not compatible to Date2  
}
```



# Mảng, con trỏ và cấu trúc

- Kết hợp mảng, con trỏ và cấu trúc cho phép xây dựng và sử dụng các cấu trúc dữ liệu phức tạp một cách rất linh hoạt

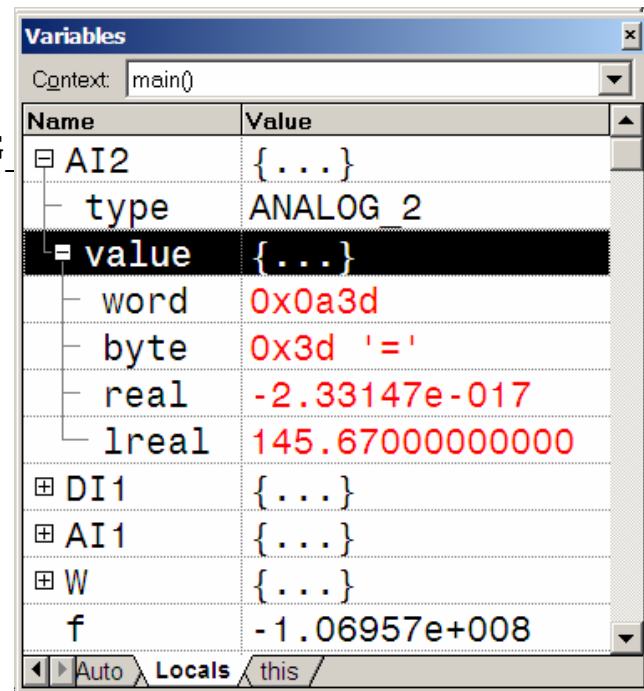
```
void main() {  
    //...  
    Date victoryDays[] = {{19, 8, 1945}, {7, 5, 1954}, {30, 4, 1975}};  
    Date saigonVictory = victoryDays[2];  
    Date *p = &saigonVictory;  
    (*p).year += 30;        // good  
    p->year -= 30;          // better  
    Student studentList[45];  
    for (int i = 0; i < 45; ++i) {  
        studentList[i].id_number = i;  
        studentList[i].birthday = yourBirthDay;  
    }  
    Student* pList = studentList;  
    while (pList < studentList + 45) {  
        pList->id_number += 4800;  
        ++pList;  
    }  
}
```

# Tóm lược về cấu trúc (struct)

- Cấu trúc (struct) được sử dụng để nhóm các dữ liệu liên quan mô tả một đối tượng, các dữ liệu có thể cùng hoặc khác kiểu
- Định nghĩa kiểu cấu trúc bằng cách khai báo **tên các biến thành viên**. Định nghĩa kiểu cấu trúc chưa phải là định nghĩa các biến cụ thể, vì thế không được đặt giá trị đầu cho các biến
- Kích cỡ của cấu trúc  $\geq$  tổng kích cỡ các thành viên
- Truy cập một biến cấu trúc thông qua tên biến, toán tử (.) và tên biến thành viên
- Các kiểu cấu trúc có thể lồng vào nhau, trong cấu trúc có thể sử dụng mảng, một mảng có thể có các phần tử là cấu trúc, v.v...
- Các biến có cùng kiểu cấu trúc có thể gán cho nhau, có thể sử dụng để khởi tạo cho nhau (khác hẳn với mảng)
- Có thể sử dụng con trỏ để truy nhập dữ liệu cấu trúc thông qua toán tử (\*) và toán tử (->)
- Hai kiểu cấu trúc có khai báo giống nhau hoàn toàn vẫn là hai kiểu cấu trúc khác nhau

## 2.4.2 Hợp nhất

```
enum SignalType {BINARY_8, BINARY_16, ANALOG_8, ANALOG_16};
union SignalValue {
    unsigned short word;
    unsigned char byte;
    float real;
    double lreal;
};
struct Signal {
    SignalType type;
    SignalValue value;
};
void main() {
    SignalValue B,W;
    B.byte = 0x01;
    W.word = 0x0101;
    unsigned char b = W.byte; // OK, the lower byte
    float f = W.real;         // meaningless
    Signal DI1 = {BINARY_8, 0x11};
    Signal AI1 = {ANALOG_1, {0}};
    Signal AI2;
    AI2.type = ANALOG_2;
    AI2.value.lreal = 145.67;
}
```



Name	Value
AI2	{...}
type	ANALOG_2
value	{...}
word	0x0a3d
byte	0x3d '='
real	-2.33147e-017
lreal	145.670000000000
DI1	{...}
AI1	{...}
W	{...}
f	-1.06957e+008

# Tóm lược về hợp nhất

- Hợp nhất (union) là một tập hợp (không có cấu trúc chặt chẽ) chứa các biến sử dụng chung ô nhớ, ở mỗi ngữ cảnh chỉ sử dụng một biến riêng biệt
- Union thường được sử dụng khi dữ liệu đầu vào có thể có kiểu khác nhau
- Các thành viên của một union không liên quan đến nhau, không cùng nhau tạo thành một thực thể thống nhất
- Kích cỡ của union bằng kích cỡ của biến lớn nhất
- Khai báo kiểu union tương tự như khai báo struct, nhưng ý nghĩa khác hẳn
- Truy nhập biến thành viên cũng tương tự như struct, có thể qua biến trực tiếp hoặc qua biến con trỏ.
- Union có thể chứa struct, struct có thể chứa union, union có thể chứa mảng, các phần tử của mảng có thể là union.

## 2.5 Điều khiển CT: phân nhánh

- Các kiểu phân nhánh
  - **if .. else**: Phân nhánh lựa chọn một hoặc hai trường hợp
  - **switch .. case**: Phân nhánh lựa chọn nhiều trường hợp
  - **break**: Lệnh nhảy kết thúc (sớm) một phạm vi
  - **return**: Lệnh nhảy và kết thúc (sớm) một hàm
  - **goto**: Lệnh nhảy tới một nhãn (**không nên dùng!**)

## 2.5.1 Cấu trúc if .. else

- Lựa chọn một trường hợp: sử dụng **if**

```
if (npoints >= 60)
    cout << "Passed";
if (npoints >= 80 && npoints <= 90) {
    grade = 'A';
    cout << grade;
}
```

- Phân nhánh hai trường hợp: sử dụng **if .. else**

```
if (npoints >= 90)
    cout << 'A';
else if (npoints >= 80)
    cout << 'B';
    else if (npoints >= 70)
        cout << 'C';
        else if (npoints >= 60)
            cout << 'D';
            else
                cout << 'F';
```

# Ví dụ: Hàm max()

```
int max1(int a, int b) {  
    int c;  
    if (a > b)  
        c = a;  
    else  
        c = b;  
    return c;  
}
```

```
int max2(int a, int b) {  
    int c = a;  
    if (a < b)  
        c = b;  
    return c;  
}
```

```
int max3(int a, int b) {  
    if (a < b)  
        a = b;  
    return a;  
}
```

```
int max4(int a, int b) {  
    if (a > b)  
        return a;  
    else  
        return b;  
}
```

```
int max5(int a, int b) {  
    if (a > b)  
        return a;  
    return b;  
}
```

```
int max6(int a, int b) {  
    return (a > b) ? a : b;  
}
```

## 2.5.2 Cấu trúc switch .. case

```
Signal input;
int i = 0;
while (i++ < 8) {
    input = readInput(i); // read from input module i
    switch (input.type) {
        case BINARY_8:
            cout << input.value.byte; break;
        case BINARY_16:
            cout << input.value.word; break;
        case ANALOG_1:
            cout << input.value.real; break;
        case ANALOG_2:
            cout << input.value.lreal; break;
        default:
            cout << "Unknown signal type";
    }
}
```



## 2.6 Điều khiển CT: vòng lặp

- Các kiểu vòng lặp trong C/C++
  - **while** (*condition*) { }
  - **do** { } **while** (*condition*)
  - **for** (*init;condition;post\_action*) { }
- Vòng lặp có thể thực hiện với **if..else** + **goto**, song không bao giờ nên như vậy
- Ứng dụng vòng lặp chủ yếu trong làm việc với mảng và các cấu trúc dữ liệu tổng quát khác => truy nhập qua biến mảng + chỉ số, qua con trỏ hoặc qua **iterator** (sẽ đề cập sau này)

## 2.6.1 Cấu trúc while..

```
#include <iostream.h>
void main() {
    char input[32];
    cout << "\nEnter your full name:";
    cin.getline(input, 31);
    short nLetters=0, nSpaces=0;
    short i=0;
    while (input[i] != 0) {
        if (input[i] == ' ')
            ++nSpaces;
        else
            ++nLetters;
        ++i;
    }
    cout << "\nYour name has " << nLetters << " letters";
    cout << "\nYou have " << nSpaces - 1 << " middle name";
    cin >> i;
}
```

# Cấu trúc while: Biểu thức điều kiện

```
#include <iostream.h>
void main() {
    char input[32], family_name[16]={0};
    cout << "\nEnter your full name:";
    cin.getline(input,31);
    short i=0;
    while (input[i] != 0) {
        if (input[i] == ' ') break;
        family_name[i]= input[i];
        ++i;
    }
    cout << "\nYour family name is " << family_name;
    cin >> i;
}
```

## 2.6.2 Cấu trúc do while...

```
#include <iostream.h>
void main() {
    char input[32], family_name[16]={0};
    short i;
    do {
        cout << "\nEnter your full name:";
        cin.getline(input,31);
        i = 0;
        while (input[i] != 0 && input[i] != ' ') {
            family_name[i]= input[i];
            ++i;
        }
        cout << "\nYour family name is " << family_name;
        cout << "\nDo you want to continue? (Y/N):";
        cin >> i;
    } while (i == 'Y' || i == 'N')
}
```

## 2.6.3 Cấu trúc for ..

```
short i =0;
while (input[i] != 0)
{
    if (input[i]==' ')
        ++nSpaces;
    else
        ++nLetters;
    ++i;
}
```

```
for (short i=0; input[i] !=0; ++i)
{
    if (input[i] == ' ')
        ++nSpaces;
    else
        ++nLetters;
}
```

```
short i=0;
for (; input[i] != 0;)
{
    if (input[i]==' ')
        ++nSpaces;
    else
        ++nLetters;
    ++i;
}
```

```
short i=0;
for (; input[i] !=0; ++i)
{
    if (input[i] == ' ')
        ++nSpaces;
    else
        ++nLetters;
}
```

# Tóm lược các cấu trúc vòng lặp

- Các cấu trúc vòng lặp **while** và **for** tương tự như nhau, thực ra ta chỉ cần một trong hai
- Cấu trúc **do...while** tuy có ý nghĩa khác một chút, song cũng có thể chuyển về cấu trúc **while** hoặc **for**
- Các cấu trúc có thể lồng vào nhau tương đối tự do, tuy nhiên tránh lồng quá nhiều để còn dễ bao quát, khi cần có thể phân hoạch lại thành hàm
- Điều khiển vòng lặp có thể nằm trực tiếp trên điều kiện, hoặc có thể kết hợp bên trong vòng lặp với các lệnh **if...else** và **break, return**
- Thận trọng trong kiểm tra điều kiện vòng lặp (chỉ số mảng, con trỏ, ...)

# Luyện tập ở nhà theo sườn bài giảng

- Tập tạo dự án mới với Visual C++
- Tập viết một chương trình bằng C (đặt đuôi \*.c):
  - tập khai báo các loại biến, sử dụng các kiểu dữ liệu cơ bản
  - tập sử dụng các phép toán đã học
  - sử dụng toán tử **sizeof** để tìm kích cỡ các kiểu dữ liệu, in kết quả ra màn hình
  - biên dịch, chạy thử và tìm lỗi
  - tập sử dụng công cụ debugger
  - đổi đuôi file thành \*.cpp và thử lại
- Tập viết một chương trình bằng C/C++ khác để tìm hiểu:
  - Cách khai báo và sử dụng kiểu hằng, kiểu liệt kê, kiểu con trỏ, kiểu mảng, kiểu tham chiếu (C++), kiểu cấu trúc
  - bản chất của con trỏ và quan hệ với kiểu mảng

# Bài tập về nhà cho chương 2

1. Viết một chương trình bằng C, thực hiện lần lượt các chức năng sau đây:
  - yêu cầu người sử dụng nhập một số nguyên lớn hơn 0
  - phân tích số nguyên đó thành hàng đơn vị, hàng chục, hàng trăm, v.v... và in kết quả lần lượt ra màn hình.
  - hỏi người sử dụng có yêu cầu tiếp tục hay không, nếu có yêu cầu thì lặp lại
2. Chuyển chương trình thành C++ và đơn giản hóa các câu lệnh vào-ra bằng cách sử dụng thư viện `<iostream.h>`
3. Dựa vào kiểu Date trong bài giảng, viết một chương trình cho phép người sử dụng nhập số liệu cho một ngày, và sau đó:
  - a) Kiểm tra các số liệu ngày, tháng và năm có hợp lệ hay không
  - b) Kiểm tra xem ngày đó có phải là một ngày lễ trong năm hay không
  - c) Xác định ngày tiếp theo là ngày nào
  - d) In các kết quả thông báo ra màn hình



# Bài tập lớn 1 (tuần 1-6: Lập trình cấu trúc)

1. Xây dựng một chương trình có chức năng tạo tín hiệu theo yêu cầu người sử dụng về dạng tín hiệu (bậc thang, tín hiệu dốc, xung vuông, hình sin hoặc ồn trắng), tham số của tín hiệu (tùy theo dạng tín hiệu chọn như biên độ, tần số, độ dốc, độ rộng xung,...). Yêu cầu người sử dụng nhập khoảng thời gian cần tạo giá trị tín hiệu cùng thời gian trích mẫu, sau đó ghi các giá trị gián đoạn của tín hiệu ra một file với tên do người sử dụng nhập.

Gợi ý: sử dụng thư viện `<fstream.h>` cho việc thao tác với file.

2. Xây dựng một chương trình để tính tích phân của tín hiệu (hay tính diện tích dưới đường cong) bằng phương pháp xấp xỉ hình thang với các giá trị gián đoạn của tín hiệu đưa vào từ file tạo ra theo chương trình 1.
3. Suy nghĩ phân hoạch chương trình 1 và 2 thành các hàm đưa vào thư viện. Viết lại các chương trình đó theo thiết kế mới.

# Chỉ dẫn về thực hiện bài tập lớn

- Bài tập lớn có thể thực hiện riêng hoặc theo nhóm tự chọn (tối đa 3 người/nhóm)
- Bài tập lớn 1 nộp vào cuối tuần 7, bao gồm:
  - Mô tả theo mẫu trên ít nhất 1 trang giấy về các tư tưởng phân tích, thiết kế và thực thi.
  - Toàn bộ thư mục dự án (file dự án, mã nguồn và chương trình chạy) cần nén lại dưới dạng \*.zip và gửi về địa chỉ email của giáo viên: [luuhongviet@gmail.com](mailto:luuhongviet@gmail.com). Qui định tên file zip: bắt đầu bằng “P1\_”, tiếp theo là tên đầy đủ của người đại diện nhóm, ví dụ “P1\_NguyenVanA.zip”. Lưu ý trước khi nén cần xóa tất cả các file phụ trong thư mục “Debug”, chỉ trừ file \*.exe.
- Hoàn thành bài tập lớn không những là điều kiện dự thi học kỳ, mà điểm bài tập lớn còn được tính vào điểm cuối học kỳ theo một hệ số thích hợp