

# Kỹ thuật lập trình

## Con trỏ - 01

Nguyễn Trọng Việt

# Mục tiêu

Sau buổi học, sinh viên có thể:

- Liệt kê loại vùng bộ nhớ
- Giải thích call stack
- Nêu nguyên tắc quản lý bộ nhớ
- Nêu khái niệm con trỏ trong C/C++
- Liệt kê tất cả đặc điểm của con trỏ
- Khai báo và khởi tạo con trỏ
- Sử dụng toán tử lấy địa chỉ biến, truy xuất nội dung biến, truy xuất trường dữ liệu, di chuyển con trỏ
- Sử dụng con trỏ trong tham số, trị trả về của hàm

# Nội dung

- Bộ nhớ
- Con trỏ cơ bản

# Nội dung

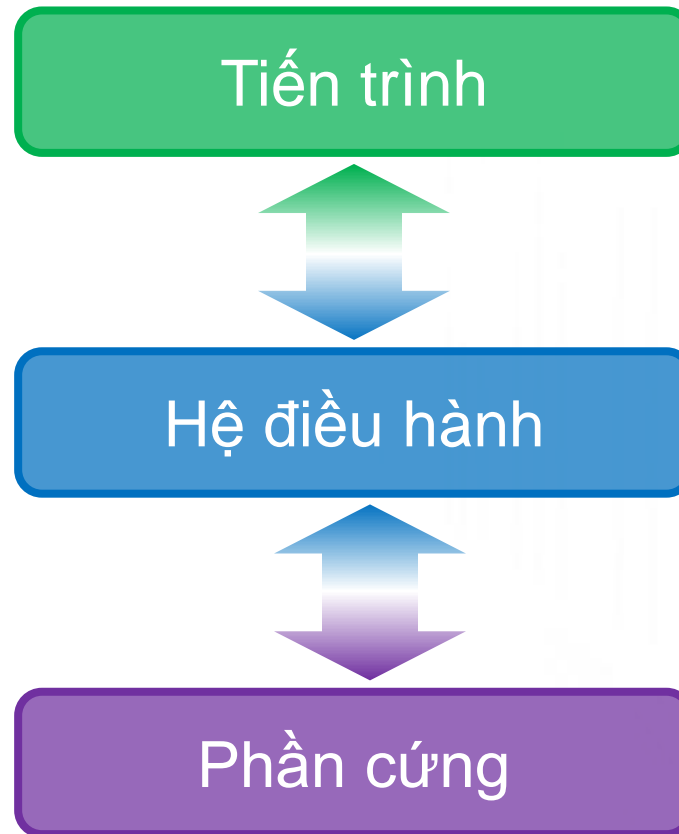
- Bộ nhớ
- Con trỏ cơ bản

**Tiến trình**  
**Bộ nhớ ảo**  
**Loại vùng nhớ**  
**Call stack**  
**Quản lý bộ nhớ**  
**Địa chỉ bộ nhớ**

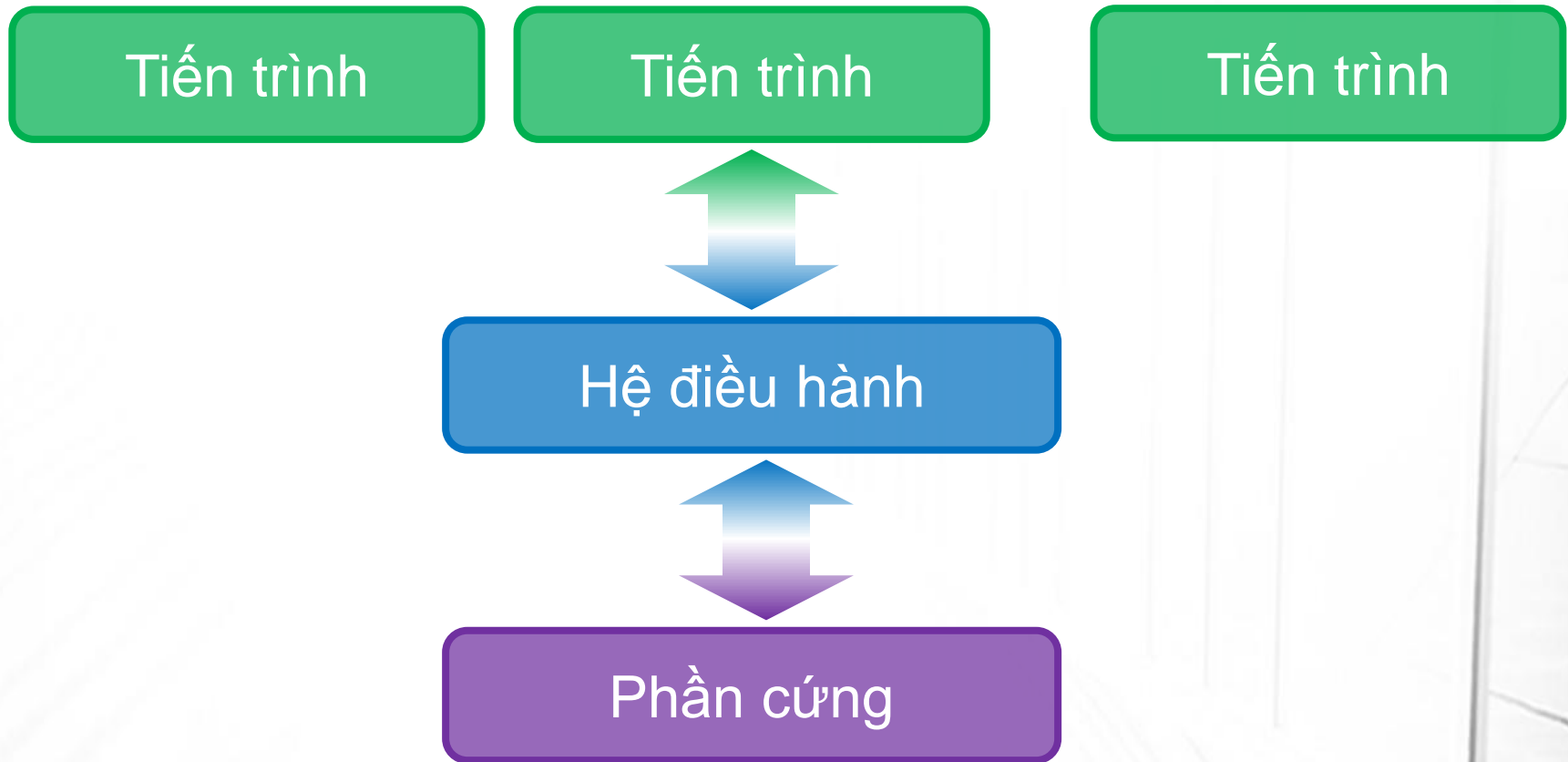
# Tiến trình

- Process
- Tiến trình là **thể hiện (instance)** của một **chương trình phần mềm** được **hệ điều hành** thực thi
  - Chương trình phần mềm được lưu trữ trên đĩa ở dạng file nhị phân có khả năng thực thi.
  - File thực thi chứa mã thực thi & dữ liệu để hệ điều hành thực thi chương trình.
  - Khi được kích hoạt, file thực thi được hệ điều hành cấp phát các tài nguyên cần thiết và bắt đầu chạy.
  - Hệ điều hành quản lý thể hiện của file thực thi thông qua **tiến trình**.

## Tiến trình (tiếp)

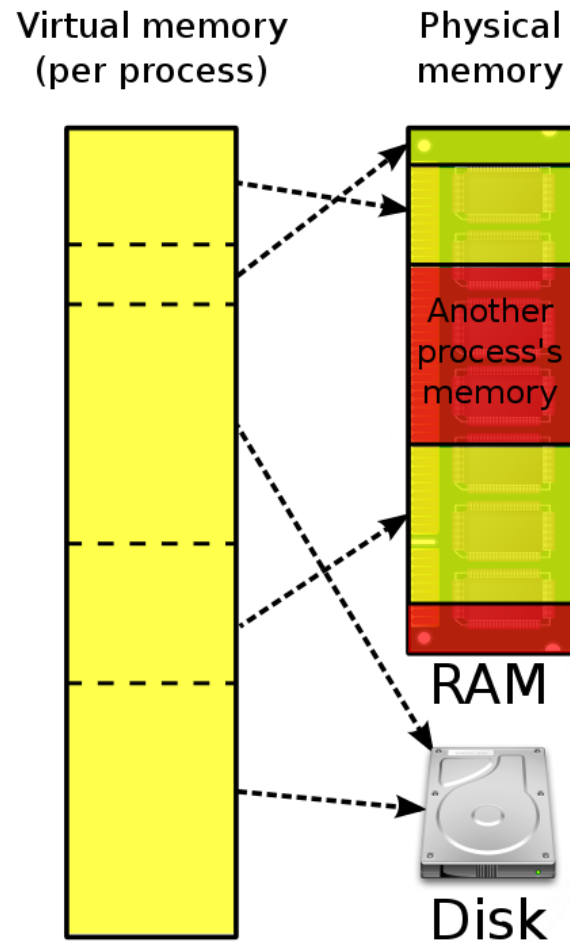


## Tiến trình (tiếp)

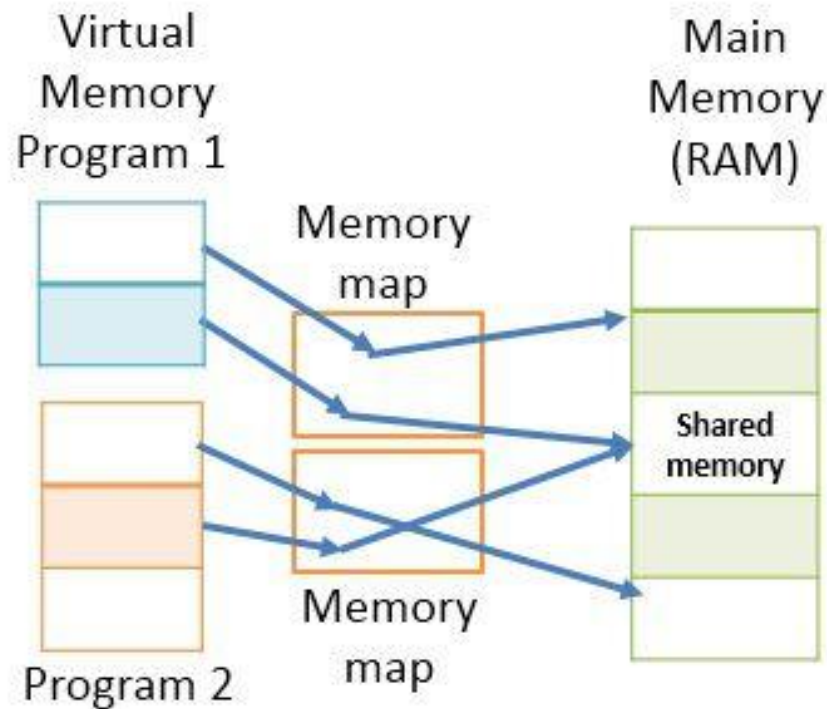




# Bộ nhớ ảo



## Bộ nhớ ảo (tiếp)

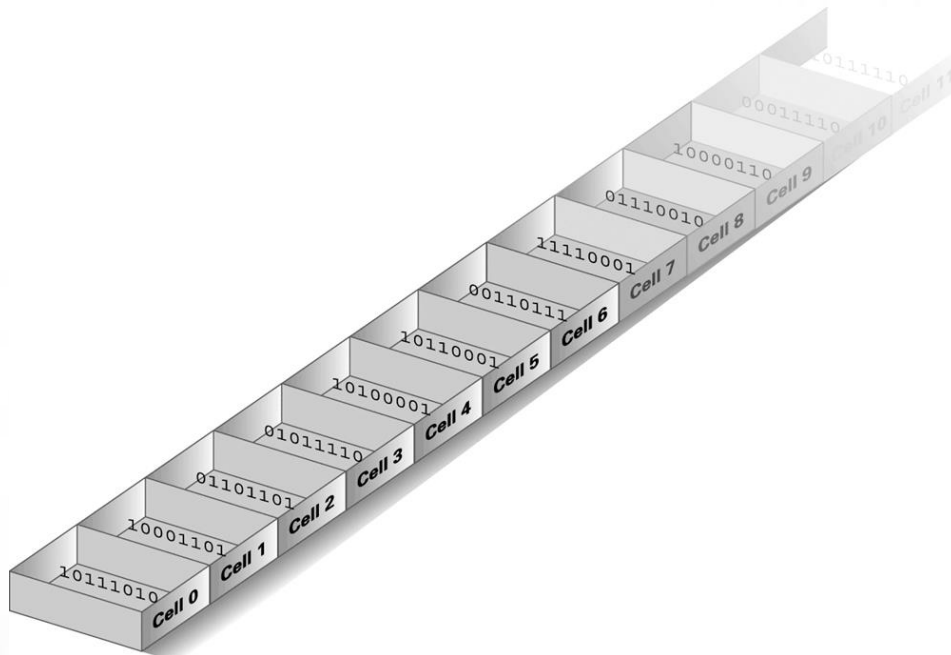


# Tế bào bộ nhớ

- Memory cell
- Là 1 đơn vị của bộ nhớ chính
- Có kích thước bằng 1 byte

## Địa chỉ bộ nhớ

- Là chuỗi số xác định duy nhất một tế bào bộ nhớ trên bộ nhớ chính máy tính
- Địa chỉ bộ nhớ được đánh liên tiếp bắt đầu từ 0



# Phân loại vùng bộ nhớ

- **Vùng nhớ Stack**

- Vùng nhớ dành cho biến cục bộ, tham số của một hàm
- Kích thước hạn chế
- Bị thu hồi tự động khi kết thúc hàm

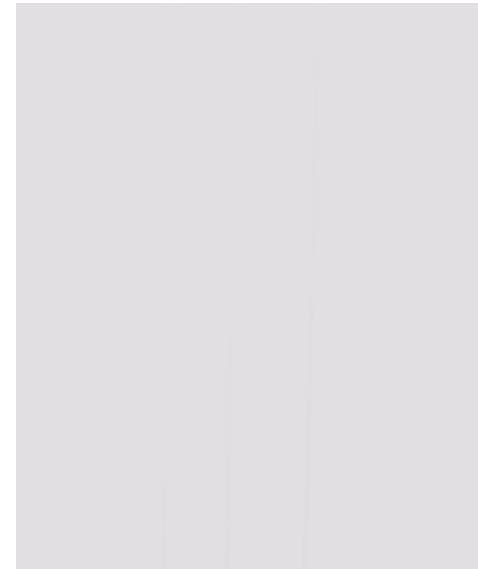
- **Vùng nhớ Heap**

- Vùng nhớ dành cho cấp phát động
- Kích thước *tự do*
- **Tiến trình tự quản lý**

# Call stack

```
void doSomething(int b)
{
    ...
}

void main()
{
    int a = 10;
    doSomething(a);
    ...
}
```



Stack

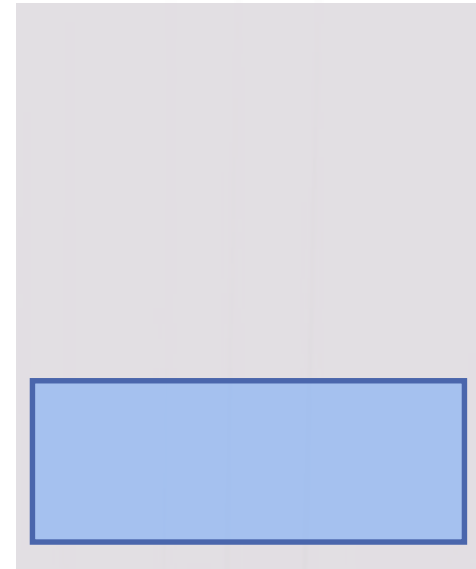
# Call stack (tiếp)

```
void doSomething(int b)
{
    ...
}

void main()
{
    int a = 10;
    doSomething(a);
    ...
}
```

main

Stack



# Call stack (tiếp)

```
void doSomething(int b)
{
    ...
}

void main()
{
    int a = 10;
    doSomething(a);
    ...
}
```

main

a = 10


Stack



# Call stack (tiếp)

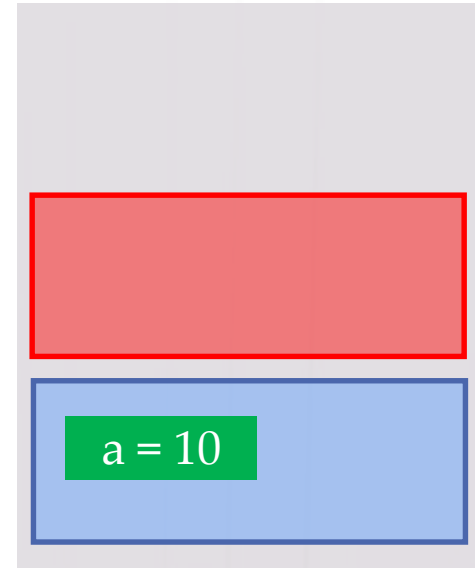
```
void doSomething(int b)
{
    ...
}

void main()
{
    int a = 10;
    doSomething(a);
    ...
}
```




doSomething

main



Stack

# Call stack (tiếp)

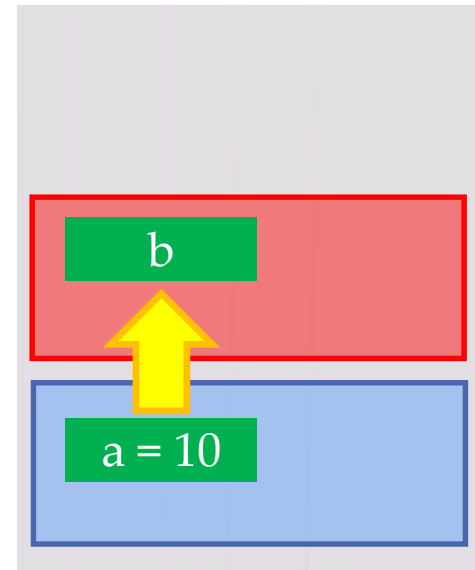


```
void doSomething(int b)
{
    ...
}

void main()
{
    int a = 10;
    doSomething(a);
    ...
}
```


doSomething

main



Stack

# Call stack (tiếp)

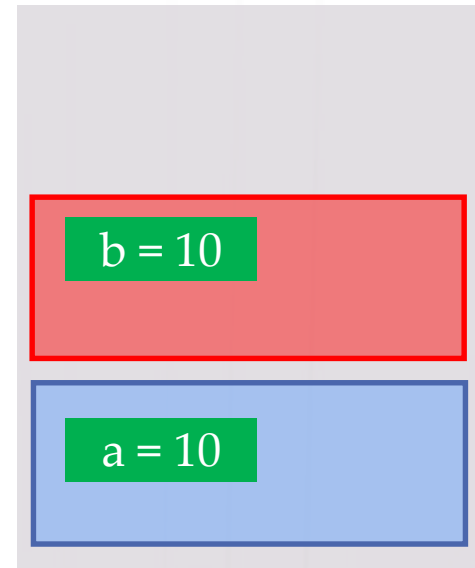


```
void doSomething(int b)
{
    ...
}

void main()
{
    int a = 10;
    doSomething(a);
    ...
}
```

doSomething

main




Stack

# Call stack (tiếp)

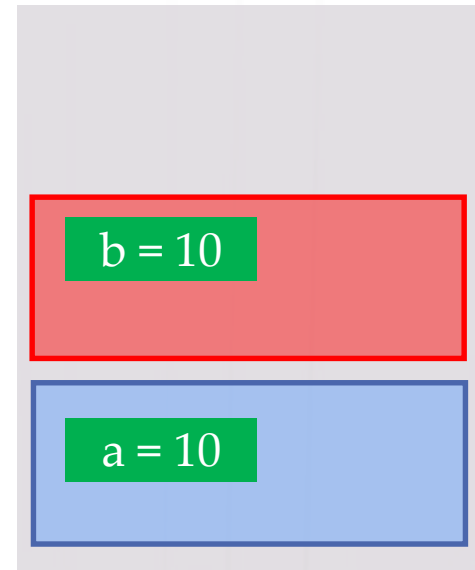
```
void doSomething(int b)
{
    ...
}

void main()
{
    int a = 10;
    doSomething(a);
    ...
}
```



doSomething

main



Stack

# Nguyên tắc quản lý bộ nhớ

## Lập trình viên

- **Vùng nhớ Stack**

- Không truy xuất biến đã bị thu hồi khi kết thúc hàm

- **Vùng nhớ Heap**

- Yêu cầu hệ thống cấp phát vùng nhớ có kích thước cho trước
- Có trách nhiệm yêu cầu hệ thống thu hồi bộ nhớ đã cấp phát
- Thao tác trong phạm vi vùng nhớ được cấp phát

## Địa chỉ bộ nhớ

- Một biến được khai báo cần có kiểu xác định
- Hệ điều hành cấp phát bộ nhớ dành cho biến dựa trên kiểu dữ liệu của biến
- Vùng nhớ dành cho biến được cấp trong Stack
- Hệ điều hành định vị vùng bộ nhớ dành cho biến bằng địa chỉ trong Stack

```
void main()  
{  
    int a = 10;  
    double b = 0.68;  
    ...  
}
```

# Toán tử lấy địa chỉ

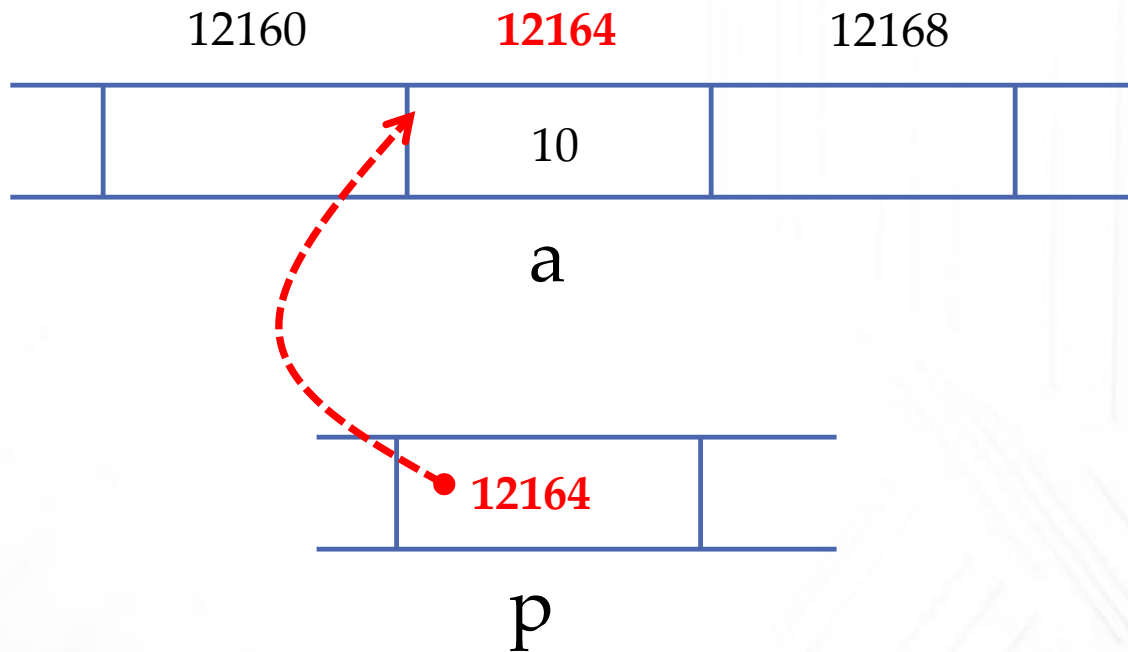
- Toán tử: **&**
  - Toán tử 1 ngôi (1 số hạng)
  - Input: biến cần lấy địa chỉ
  - Output: số *nguyên* là địa chỉ vùng nhớ của biến
- Cách sử dụng:
  - Đặt toán tử **&** trước biến muốn lấy địa chỉ

**&**<variable>

```
...  
int a = 10;  
p = &a;
```

## Toán tử lấy địa chỉ (tiếp)

```
...  
int a = 10;  
p = &a
```





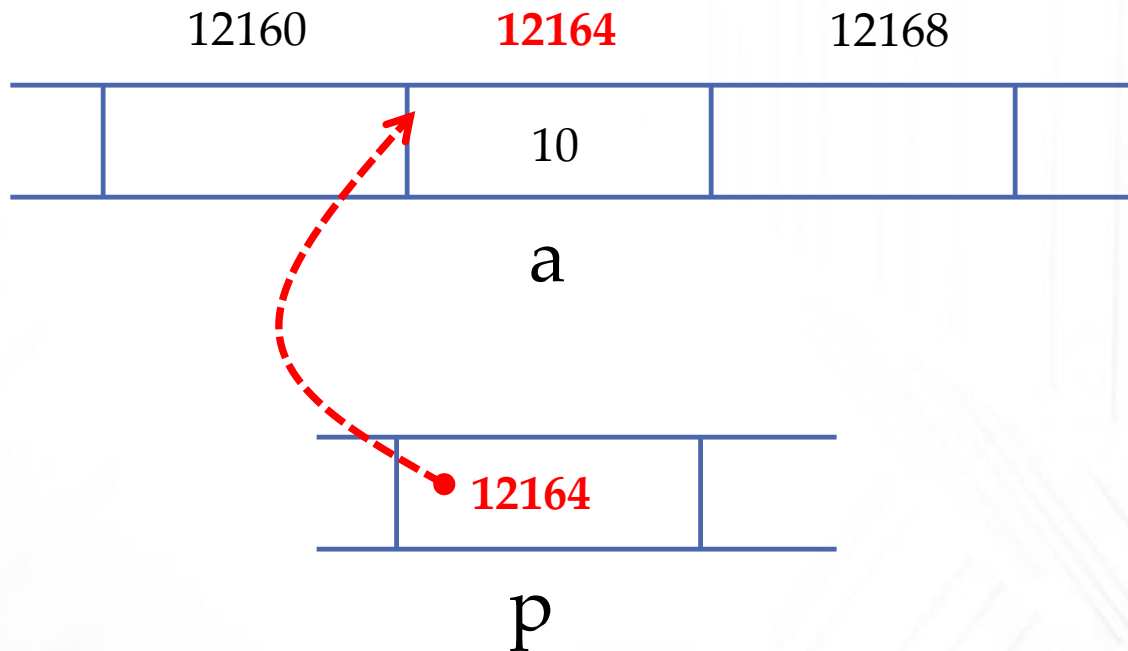
# Nội dung

- Bộ nhớ
- Con trỏ cơ bản

**Khái niệm & đặc điểm**  
**Khai báo & khởi tạo**  
**Truy xuất nội dung**  
**Số học con trỏ**  
**Sử dụng con trỏ**

# Khái niệm con trỏ

- Pointer
- Là một biến đặc biệt trong C/C++ dùng để **lưu địa chỉ vùng nhớ** của biến khác có kiểu phù hợp



# Khai báo con trỏ

- Cú pháp

```
type *name;
```

- Trong đó:

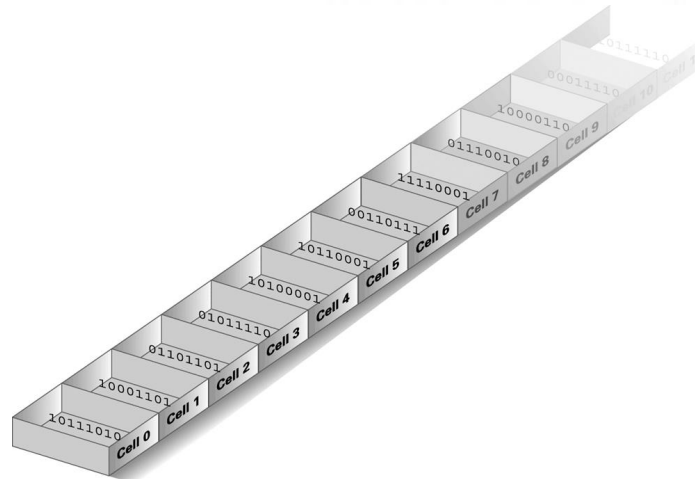
- name: tên biến con trỏ
- type: kiểu của vùng nhớ mà con trỏ sẽ giữ địa chỉ

- Ví dụ:

```
int *ip;  
double* dp;  
char * cp;
```

# Đặc điểm của con trỏ

- Vì là 1 biến nên con trỏ cũng được cấp bộ nhớ
- Kích thước vùng nhớ của con trỏ luôn cố định
- Kích thước vùng nhớ con trỏ bằng kích thước kiểu số nguyên
- Giá trị lưu trong vùng nhớ của con trỏ là 1 số nguyên, đó là địa chỉ của 1 vùng nhớ khác



# Định nghĩa kiểu con trỏ - typedef

- Cú pháp

```
typedef <type>* <type_name>;
```

- Trong đó:

- `type`: kiểu của vùng nhớ mà con trỏ sẽ giữ địa chỉ
- `type_name`: tên kiểu thay thế

- Ví dụ:

```
typedef int* pint;  
typedef float* pfloat;  
  
pint p1;  
pfloat p2;
```

# Định nghĩa kiểu con trỏ - typedef

```
struct Student
{
    char firstName[64];
    char lastName[64];
    char middleName[256];
    char id[32];
    int age;
    ...
};

typedef Student* PStudent;

Student* p1;
PStudent p2;
```

# Định nghĩa kiểu con trỏ - `using`

- Cú pháp

```
using <type_name> = <type>*;
```

- Trong đó:

- `type`: kiểu của vùng nhớ mà con trỏ sẽ giữ địa chỉ
- `type_name`: tên kiểu thay thế

- Ví dụ:

```
using pint = int*;  
using pfloat = float*;  
  
pint p1;  
pfloat p2;
```



## Định nghĩa kiểu con trỏ - using

```
struct Student
{
    char firstName[64];
    char lastName[64];
    char middleName[256];
    char id[32];
    int age;
    ...
};

using PStudent = Student*;

Student* p1;
PStudent p2;
```

# Khai báo và gán giá trị mặc định

```
int *p1, *p2;  
int* p1, p2;
```

```
int *p1;  
int *p2;
```

```
int* p1;  
int* p2;
```

```
int* p1 = NULL;  
int* p2 = NULL;
```

```
int* p1 = nullptr;  
int* p2 = nullptr;
```

## Gán giá trị cho con trỏ

- Con trỏ chỉ có thể giữ địa chỉ của vùng nhớ có kiểu phù hợp

```
int a = 10;  
int *pa = &a;           // yes  
  
float b = 34.5f;  
int *pb = &b;           // no
```

## Gán con trỏ cho con trỏ

- Gán giá trị của 1 con trỏ cho 1 con trỏ khác
- Bản chất là phép copy dữ liệu từ con trỏ này sang con trỏ kia
- 2 con trỏ sẽ giữ địa chỉ của cùng 1 vùng bộ nhớ
- 2 con trỏ cùng trỏ đến 1 vùng nhớ

```
int a = 10;  
int *pa = &a;  
int *pb = pa;  
*pb += 10;  
  
cout << a << *pa << *pb;
```

# Toán tử truy xuất nội dung vùng nhớ

- Toán tử: \*

\*<pointer>

- Toán tử 1 ngôi (1 số hạng)
- Input: biến con trỏ
- Output: *giá trị chứa trong vùng nhớ con trỏ đang trỏ đến*
- Cách sử dụng:
  - Đặt toán tử \* trước biến con trỏ

```
...  
int a = 10;  
int* p = &a;  
  
int b = *p;
```

## Toán tử truy xuất nội dung vùng nhớ (tiếp)

```
int a;  
int b;  
  
int* p = &a;  
*p = 10;  
  
p = &b;  
*p = 20;  
  
cout << "a: " << a << "\n";  
cout << "b: " << b << "\n";
```

# Toán tử truy xuất trường dữ liệu

- Toán tử: .
  - Toán tử 2 ngôi (2 số hạng)
  - Input:
    - Biến kiểu cấu trúc (structure)
    - Tên trường dữ liệu cần lấy giá trị
  - Output:
    - Giá trị của trường dữ liệu
- Cách sử dụng:
  - Đặt toán tử . giữa biến và tên trường dữ liệu

```
<variable> . <field_name>
```

## Toán tử truy xuất trường dữ liệu (tiếp)

```
struct Student
{
    char name[32];
    int age;
};

...

Student student;

int his_age = student.age;
```



# Toán tử truy xuất trường dữ liệu (tiếp)

- Toán tử: **->**

`<pointer>-><field_name>`

- Toán tử 2 ngôi (2 số hạng)
- Input:
  - Con trỏ đến biến kiểu cấu trúc
  - Tên trường dữ liệu cần lấy giá trị
- Output:
  - Giá trị của trường dữ liệu
- Cách sử dụng:
  - Đặt toán tử **->** giữa con trỏ và tên trường dữ liệu

## Toán tử truy xuất trường dữ liệu (tiếp)

```
struct Student
{
    char name[32];
    int age;
};

...

Student student;
Student* p = &student;

int his_age = p->age;
int someone_age = (*p).age;
```

## Số học con trỏ

- Chỉ hỗ trợ phép cộng và trừ
- Là phép di chuyển con trỏ về trước và sau
- Độ lớn khoảng di chuyển phụ thuộc kiểu dữ liệu của vùng nhớ con trỏ trỏ tới

## Số học con trỏ (tiếp)

```
char* pc;  
int* pi;  
long* pl;  
...  
pc = pc + 1;  
pi = pi + 1;  
pl = pl + 1;
```

```
char* pc;  
int* pi;  
long* pl;  
...  
++pc;  
++pi;  
++pl;
```

- Giả sử pc, pi và pl lần lượt trỏ đến các vùng nhớ tại địa chỉ 3000, 4000, 5000
- Sau phép cộng, pc, pi và pl trỏ đến vùng nhớ có địa chỉ bao nhiêu ?

## Số học con trỏ (tiếp)

```
int a = 5;  
int b = 10;  
int c = 15;  
  
int* p = &a;  
int* q = &b;
```

- Giả sử vùng nhớ dành cho a, b và c liên tiếp nhau

1	<code>*p++;</code>	
2	<code>*++p;</code>	
3	<code>++*p;</code>	
4	<code>(*p)++;</code>	
5	<b><code>*p++ = *q++;</code></b>	

# Độ ưu tiên toán tử

Độ ưu tiên	Toán tử	Thứ tự thực hiện
1	::	Trái -> phải
2	() [] . -> a++ a--	Trái -> phải
3	++a --a +a -a ! ~ *a &a	Phải -> trái
4	* / %	Trái -> phải
5	+ -	Trái -> phải
6	>> <<	Trái -> phải
7	< <= > >=	Trái -> phải
8	== !=	Trái -> phải
...	...	...

# Số học con trỏ (tiếp)

- Phong cách lập trình:
  - Mỗi line thực hiện đúng 1 tác vụ
  - Code đơn giản để dễ đọc, dễ sửa
  - Code đơn giản để tránh bug

```
int a = 5;  
int b = 10;  
int c = 15;
```

```
int* p = &a;  
int* q = &b;
```

```
*p++ = *q++;
```



```
int a = 5;  
int b = 10;  
int c = 15;
```

```
int* p = &a;  
int* q = &b;
```

```
*p = *q;  
p++;  
q++;
```

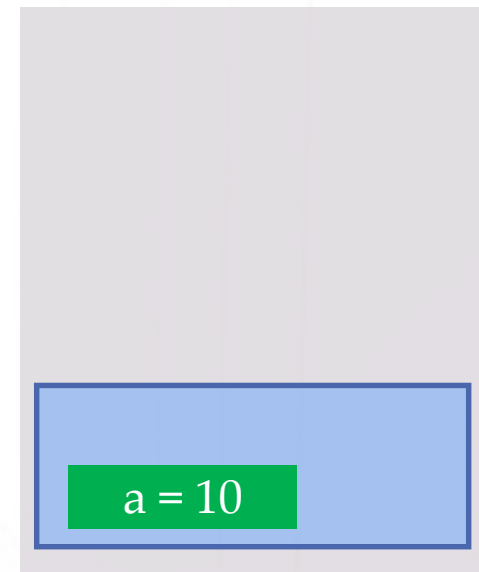
# Tham số con trỏ

- Con trỏ có thể dùng làm tham số cho 1 hàm

```
void doSomething(int* p)
{
    ...
}

void main()
{
    int a = 10;
    doSomething(&a);
    ...
}
```

main



Stack

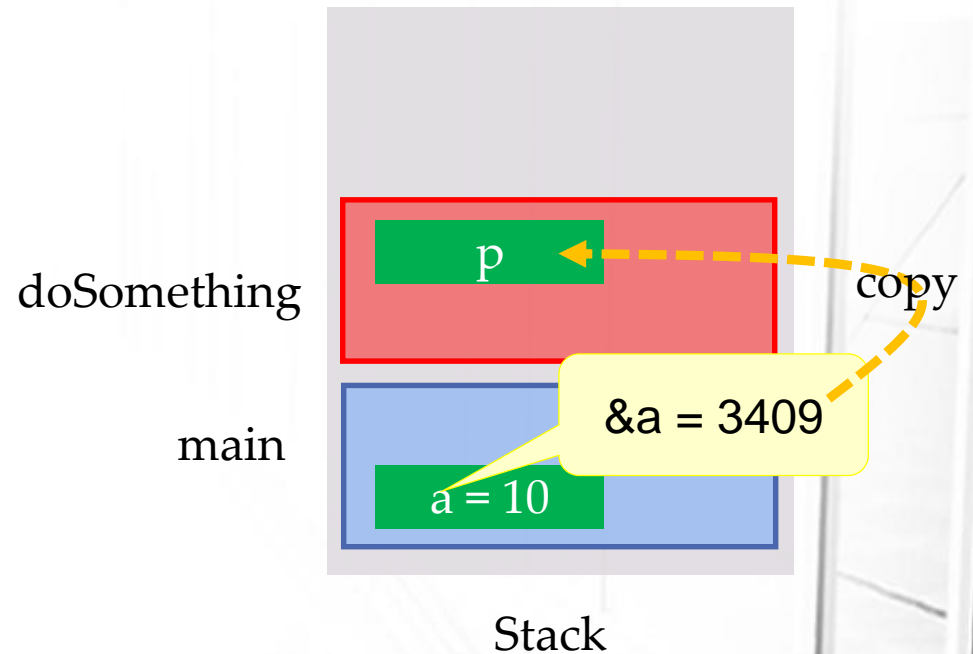


# Tham số con trỏ (tiếp)

- Con trỏ có thể dùng làm tham số cho 1 hàm

```
void doSomething(int* p)
{
    ...
}

void main()
{
    int a = 10;
    doSomething(&a);
    ...
}
```



# Tham số con trỏ (tiếp)

- Con trỏ có thể dùng làm tham số cho 1 hàm

```
void doSomething(int* p)
{
    ...
}

void main()
{
    int a = 10;
    int* q = &a;
    doSomething(q);
    ...
}
```

&a = 3409

main

a = 10

q = 3409

Stack

# Tham số con trỏ (tiếp)

- Con trỏ có thể dùng làm tham số cho 1 hàm

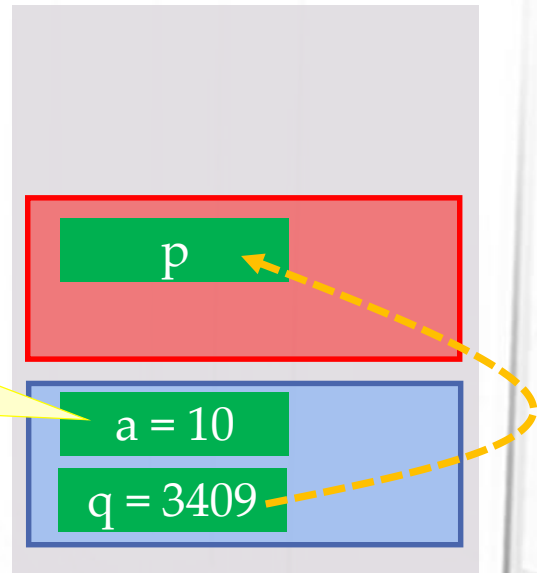
```
void doSomething(int* p)
{
    ...
}

void main()
{
    int a = 10;
    int* q = &a;
    doSomething(q);
    ...
}
```

doSomething

&a = 3409

main



Stack

# Tham số con trỏ (tiếp)

- Con trỏ có thể dùng làm tham số cho 1 hàm

```
void doSomething(int* p)
{
    ...
}

void main()
{
    int a = 10;
    int* q = &a;
    doSomething(q);
    ...
}
```

doSomething

&a = 3409

main

p = 3409

a = 10

q = 3409

Stack

## Tham số con trỏ (tiếp)

```
void doSomething(int* p)
{
    ...
}

void main()
{
    int a = 10;
    int* q = &a;

    doSomething(&a);
    doSomething(q);
    ...
}
```

- Bản chất là copy giá trị:
  - Đối số là biến: Địa chỉ của đối số
  - Đối số là con trỏ: Giá trị của đối số

# Trả về con trỏ

- Con trỏ có thể được hàm trả về

```
int* doSomething()  
{  
    int a = 10;  
    int* p = &a;  
  
    return p;  
}  
  
void main()  
{  
    int* p = doSomething();  
    int a = 10 + *p;  
    ...  
}
```

# Đánh giá đạt mục tiêu

Liệu sau bài học, sinh viên có thể:

- Liệt kê phân loại vùng bộ nhớ
- Giải thích call stack
- Nêu nguyên tắc quản lý bộ nhớ
- Nêu khái niệm con trỏ trong C/C++
- Liệt kê tất cả đặc điểm của con trỏ
- Khai báo và khởi tạo con trỏ
- Sử dụng toán tử lấy địa chỉ biến, truy xuất nội dung biến, truy xuất trường dữ liệu, di chuyển con trỏ
- Sử dụng con trỏ trong tham số, trị trả về của hàm