

# Kỹ thuật lập trình

## Con trỏ - 03

Nguyễn Trọng Việt

# Mục tiêu

Sau buổi học, sinh viên có thể:

- Sử dụng con trỏ hằng, hằng con trỏ, con trỏ void
- Sử dụng con trỏ đa cấp
- Sử dụng con trỏ đến *mảng cố định*
- Khái báo và sử dụng con trỏ hàm

# Nội dung

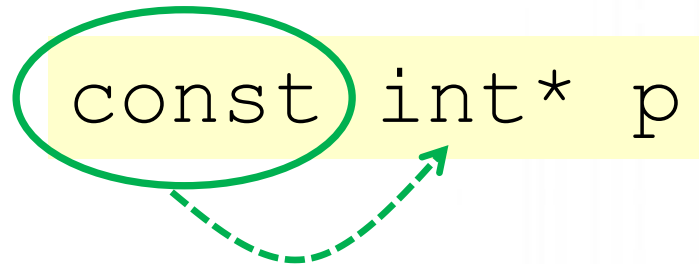
- Con trỏ & hằng
- Con trỏ void
- Con trỏ trỏ con trỏ
- Con trỏ đến mảng kích thước cố định
- Con trỏ hàm

# Nội dụng

- Con trở & hằng
- Con trở void
- Con trở trở con trở
- Con trở đến mảng kích thước cố định
- Con trở hàm

## Con trỏ hằng

- Chỉ được phép đọc (read-only) nội dung vùng nhớ được trỏ mà không được cập nhật



```
const int* p
```

- `const` đứng liền trước thứ gì sẽ *biến thứ đó thành hằng số !!*

## Con trỏ hằng (tiếp)

```
const int* p
```

Về mặt cú pháp:

- p chỉ có thể giữ địa chỉ của vùng nhớ có kiểu là `const int`
- Con trỏ hằng = con trỏ trỏ đến hằng

Về mặt thực thi:

- p có thể giữ địa chỉ của bất kì vùng nhớ nào có kiểu `int`
- p chỉ có thể đọc nội dung của vùng nhớ đó
- Con trỏ hằng = ...

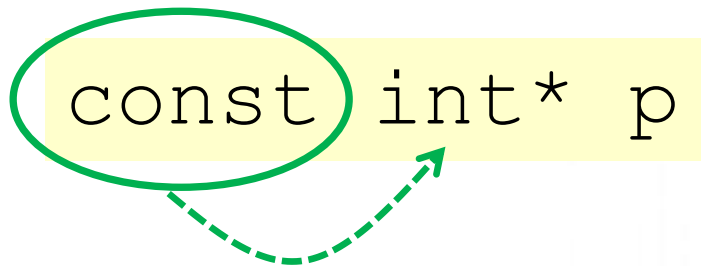
## Con trỏ hằng (tiếp)

```
int x;  
int y = 10;  
const int z = 100;  
  
const int * p1 = &y;  
const int * p2 = &z;  
  
x = *p2;  
*p1 = x; F  
int* p3 = p1;
```

Nhận xét:

- `&y` có kiểu là `int*` trong khi `p1` là `const int*`: chuyển kiểu ngầm định
- `p1` là `const int*` trong khi `p3` là `int*`: bất khả chuyển

## Con trỏ hằng (tiếp)



```
const int* p
```

Tóm lại:

- p là biến, không phải hằng
- const chỉ áp dụng cho nội dung vùng nhớ được trỏ khi truy xuất từ con trỏ
- p có thể trỏ đến vùng nhớ khác



## Con trỏ hằng (tiếp)

- Ứng dụng: truyền đối số của những cấu trúc dữ liệu đồ sộ cho hàm

```
struct Course
{
    char name[256];
    int semester;
    bool passed;
    float score;
};

struct Student
{
    char firstName[128];
    char lastName[128];
    char midName[256];
    char id[32];
    Course* courses;
    int nCourse;
    float averagedScore;
};
```

```
bool willGraduate(Student p)
{
    ...
}

void main()
{
    Student hoang;

    // first year
    // second year
    ...
    if ( willGraduate(hoang) ) {
        printf("He will graduate\n");
    }
    else {
        printf("God knows\n");
    }
}
```

## Con trỏ hằng (tiếp)

- Ứng dụng: truyền đối số của những cấu trúc dữ liệu đồ sộ cho hàm

```
struct Course
{
    char name[256];
    int semester;
    bool passed;
    float score;
};

struct Student
{
    char firstName[128];
    char lastName[128];
    char midName[256];
    char id[32];
    Course* courses;
    int nCourse;
    float averagedScore;
};
```

```
bool willGraduate(const Student* p)
{
    ...
}

void main()
{
    Student hoang;

    // first year
    // second year
    ...
    if ( willGraduate(&hoang) ) {
        printf("He will graduate\n");
    }
    else {
        printf("God knows\n");
    }
}
```

## Con tr  h ng (ti p)

```
void printAll(const int* start, const int* stop)
{
    const int* current = start;
    while (current != stop)
    {
        cout << *current << '\n';
        ++current;
    }
}

int main ()
{
    int numbers[] = {10, 20, 30};
    printAll(cnumbers, numbers + 3 );

    return 0;
}
```

# Hằng con trỏ

- Con trỏ chỉ có thể giữ địa chỉ của duy nhất 1 vùng nhớ

```
int a;  
int* const p = &a;
```

- `const` đứng liền trước thứ gì sẽ biến thứ đó thành hằng số !!

Nhận xét:

- Buộc phải gán giá ngay lập tức sau khi khai báo
- Chỉ trỏ đến duy nhất 1 nơi
- Có thể sửa đổi nội dung nơi được trỏ tới

# Con trỏ & hằng

```
int x;
```

```
int* p1 = &x;
```

```
const int* p2 = &x;
```

```
int* const p3 = &x;
```

```
const int* const p4 = &x;
```

## Con trỏ & hằng (tiếp)

```
const int* p2a = &x;  
int const* p2b = &x;
```

Các bạn có nhận thấy gì bất thường không ?

# Nội dung

- Con trỏ & hằng
- Con trỏ void
- Con trỏ trỏ con trỏ
- Con trỏ đến mảng kích thước cố định
- Con trỏ hàm

# Con trỏ void

- Trong C/C++, void biểu diễn kiểu *vắng mặt* – không định kiểu
- Con trỏ void là con trỏ trỏ đến vùng nhớ không định kiểu
  - Không xác định được độ lớn vùng nhớ được trỏ đến
  - Không thể truy xuất nội dung vùng nhớ được trỏ đến

```
int a = 100;  
double b = 10.123;  
  
void* p = nullptr;  
  
p = &a;  
p = &b;
```



## Con trỏ void (tiếp)

- Ưu điểm:
  - Cực kì mềm dẻo: có thể trỏ đến vùng nhớ có kiểu bất kì
- Khuyết điểm:
  - Không thể truy xuất trực tiếp nội dung vùng nhớ được trỏ
- Ứng dụng:
  - Kiểu của tham số cho hàm thư viện
  - Kiểu trả về của hàm thư viện

## Con tr  void (ti p)

```
void increase(void* data, int psize)
{
    if ( psize == sizeof(char) ){
        char* pchar = (char*)data;
        ++(*pchar);
        return;
    }

    if (psize == sizeof(int) ){
        int* pint = (int*)data;
        ++(*pint);
    }
}

void main ()
{
    char a = 'x';
    int b = 1602;
    increase (&a, sizeof(a));
    increase (&b, sizeof(b));
    cout << a << ", " << b << '\n';
}
```

# Nội dụng

- Con trở & hằng
- Con trở void
- Con trở trở con trở
- Con trở đến mảng kích thước cố định
- Con trở hàm

# Con trỏ trỏ đến con trỏ

- Con trỏ là biến
  - Biến cũng chiếm vùng nhớ để lưu giá trị
  - Có thể lấy địa chỉ của biến con trỏ
  - Gán địa chỉ vùng nhớ của con trỏ cho 1 con trỏ khác
- Con trỏ trỏ đến con trỏ hay con trỏ đa cấp

## Con trỏ trỏ đến con trỏ (tiếp)

```
int* * p;
```

Trỏ đến vùng nhớ kiểu `int`

Trỏ đến vùng nhớ kiểu `int*`

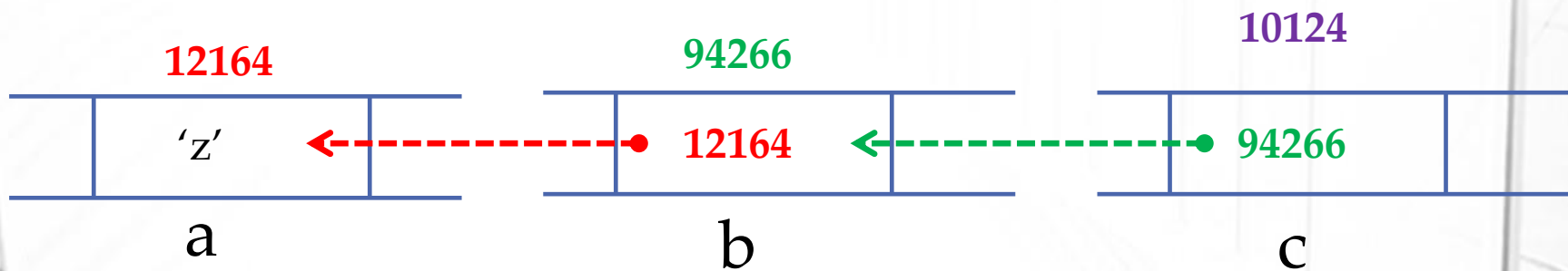
Nhận xét:

- Bản thân `p` cũng có địa chỉ
- Con trỏ 1 cấp hay n cấp đều có kích thước như nhau

## Con trỏ trỏ đến con trỏ (tiếp)

```
char a;  
char* b;  
char** c;
```

```
a = 'z';  
b = &a;  
c = &b;
```



# Nội dụng

- Con trỏ & hằng
- Con trỏ void
- Con trỏ trỏ con trỏ
- Con trỏ đến mảng kích thước cố định
- Con trỏ hàm

## Con trỏ đến mảng có kích thước cố định

```
int x[5];  
int y[20];  
  
int* p = nullptr;  
p = x;  
p = y;
```

- Khai báo thế nào để `p` trỏ đến mảng *nguyên miết* với đúng 5 phần tử kiểu `int` ?

```
int x[5];  
int y[20];  
  
int* p[5] = nullptr;  
p = x;
```



# Con trỏ đến mảng có kích thước cố định

- Khai báo

```
int x[5];  
int y[20];  
  
int (*p)[5] = nullptr;  
p = &x;
```

- p trỏ đến *nguyên miến* bộ nhớ gồm 5 phần tử kiểu int
- p phải giữ địa chỉ biến mảng thay vì biến mảng

# Con trỏ đến mảng có kích thước cố định

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

- Độ ưu tiên của các kí hiệu trong khai báo biến tuân theo độ ưu tiên của các toán tử tương ứng

# Con trỏ đến mảng có kích thước cố định

- Diễn giải

```
int *p1[5];  
int (*p2)[5];
```

# Khai báo kiểu dữ liệu

```
typedef int (*PFixedSizeArray) [5];
```

```
int x[5];  
int y[10];
```

```
PFixedSizeArray p1 = &x;  
PFixedSizeArray p2 = &y;
```

```
using PFixedSizeArray = int (*) [5];
```

```
int x[5];  
int y[10];
```

```
PFixedSizeArray p1 = &x;
```

# Truy xuất phần tử

- Truy xuất nội dung vùng nhớ đang được trỏ
- Dùng số học con trỏ để đi đến phần tử muốn truy xuất

```
using PFixedSizeArray = int (*)[5];  
  
int x[5] = {1, 2, 3, 4, 5};  
  
PFixedSizeArray p = &x;
```

- Truy xuất phần tử 1:

```
cout << (*p)[1];
```

Hay ???

```
cout << *p[1];
```

# Con trỏ đến mảng có kích thước cố định

- Phân biệt

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

```
// p1, p2, p3 are pointers
```

```
01. p1 = a;
```

```
02. p2 = &a;
```

```
03. p3 = &a[0];
```

# Nội dụng

- Con trở & hằng
- Con trở void
- Con trở trở con trở
- Con trở đến mảng kích thước cố định
- Con trở hàm

## Con trỏ hàm

- Mã thực thi của 1 hàm cần phải được load vào bộ nhớ để thực thi
- Vì hàm được cấp phát bộ nhớ nên nó cũng có địa chỉ
- Con trỏ giữ địa chỉ vùng nhớ dành chứa mã thực thi của 1 hàm được gọi là con trỏ hàm
- Con trỏ hàm có thể trỏ đến bất kì hàm nào có cùng signature với khai báo của con trỏ hàm



# Khai báo con trỏ hàm

- Là sự kết hợp của con trỏ và hàm
- Đặc trưng con trỏ: dấu \*
- Đặc trưng hàm:
  - Danh sách tham số (vắng mặt hoặc không)
  - Kiểu trả về

```
<return_type> (*pointer) (<param>);
```

- <return\_type>: kiểu trả về của hàm
- pointer: biến con trỏ
- <param>: danh sách tham số hàm

## Ví dụ con trỏ hàm

```
int addition(int a, int b) {  
    return a + b;  
}  
  
int subtraction(int a, int b) {  
    return a - b;  
}  
  
void main() {  
    int (*funcp)(int, int) = nullptr;  
  
    funcp = addition;  
    int c = funcp(5, 6);  
  
    funcp = subtraction;  
    int d = funcp(5, 6);  
  
    printf("%d, %d", c, d);  
}
```

# Khai báo kiểu con trỏ hàm

- Sử dụng typedef

```
typedef <return_type> (*type_name) (<param>)
```

- Sử dụng using

```
using type_name = <return_type> (*) (<param>)
```

- Ví dụ:

```
int addition(int a, int b);  
int subtraction(int a, int b);  
  
typedef int (*PFunc)(int, int);  
  
PFunc my_pointer = addition;
```

```
int addition(int a, int b);  
int subtraction(int a, int b);  
  
using PFunc = int (*)(int, int);  
  
PFunc my_pointer = addition;
```

# Tham số con trỏ hàm

```
int addition(int a, int b);
int subtraction(int a, int b);

void dump(int x, int y, int (*p)(int, int))
{
    int r = p(x, y);
    printf("result: %d\n", r);
}

void main() {
    int a = 10;
    int b = 11;

    dump(a, b, addition);
    dump(b, a, subtraction);
}
```

# Giải trí

```
01. int* x[100];  
02. int (*x)[100];  
03. int *(*x)[100];  
04. int (*x)();  
05. int (*x[100])();  
06. int *(*(*x[100])())();  
  
07. int (*F())();  
08. int *(*F())();  
09. int (*F())[];  
10. int ((*F())[])();  
11. int *(*(*F())[])();
```

## Giải trí

- Xác định số lượng phần tử mảng 1 chiều tĩnh

```
double a[] = {1.2, 2.3, 3.4, 4.5, 5.6};  
int n_ele = ...
```

# Giải trí

- Lấy con trỏ gốc từ 1 field của structure

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

Student s;
Student* ps = &s;
int* p_age = &ps->age;
...
Student* p_origin = ...p_age... // take from p_age
```

- Gợi ý: sử dụng macro `offsetof`

# Đánh giá đạt mục tiêu

Sau bài học, liệu sinh viên có thể:

- Sử dụng con trỏ hằng, hằng con trỏ, con trỏ void
- Sử dụng con trỏ đa cấp
- Sử dụng con trỏ đến mảng cố định
- Khái báo và sử dụng con trỏ hàm



# Đánh giá đạt mục tiêu

Liệu sau bài học, sinh viên có thể:

- Sử dụng con trỏ đa cấp
- Sử dụng con trỏ hằng, hằng con trỏ, con trỏ void
- Khái báo và sử dụng con trỏ hàm
- Nêu mối tương quan giữa con trỏ và mảng/chuỗi
- Cấp phát/thu hồi bộ nhớ động