LẬP TRÌNH NÂNG CAO

Bài 7+8+9: Con trỏ và bộ nhớ trong C/C++

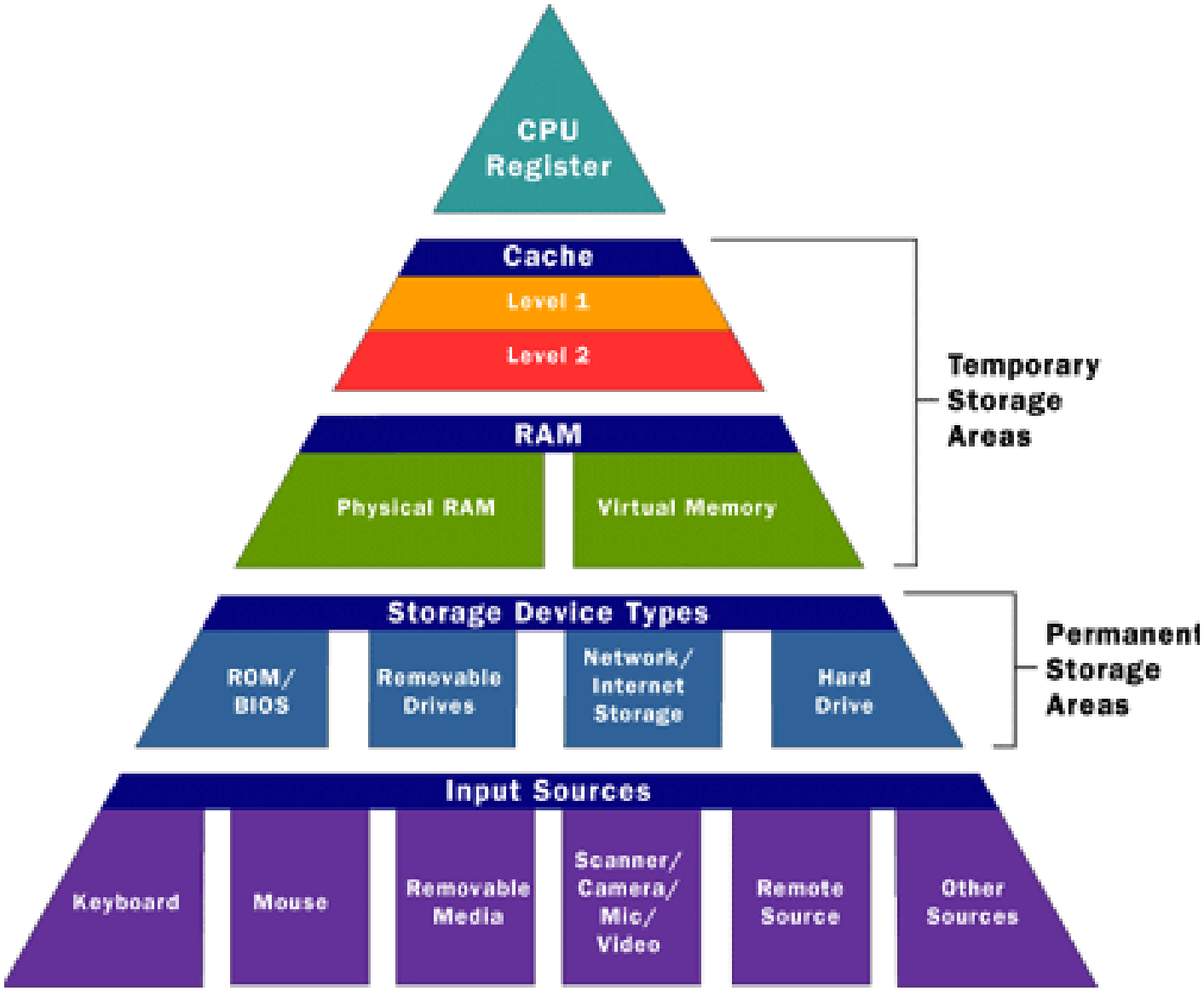
Nội dung chính

1. Bộ nhớ máy tính
2. Biến và địa chỉ của biến
3. Biến con trỏ
4. Mảng và con trỏ
5. Bộ nhớ động
6. Con trỏ hàm
7. Bài tập

Phần 1

Bộ nhớ máy tính

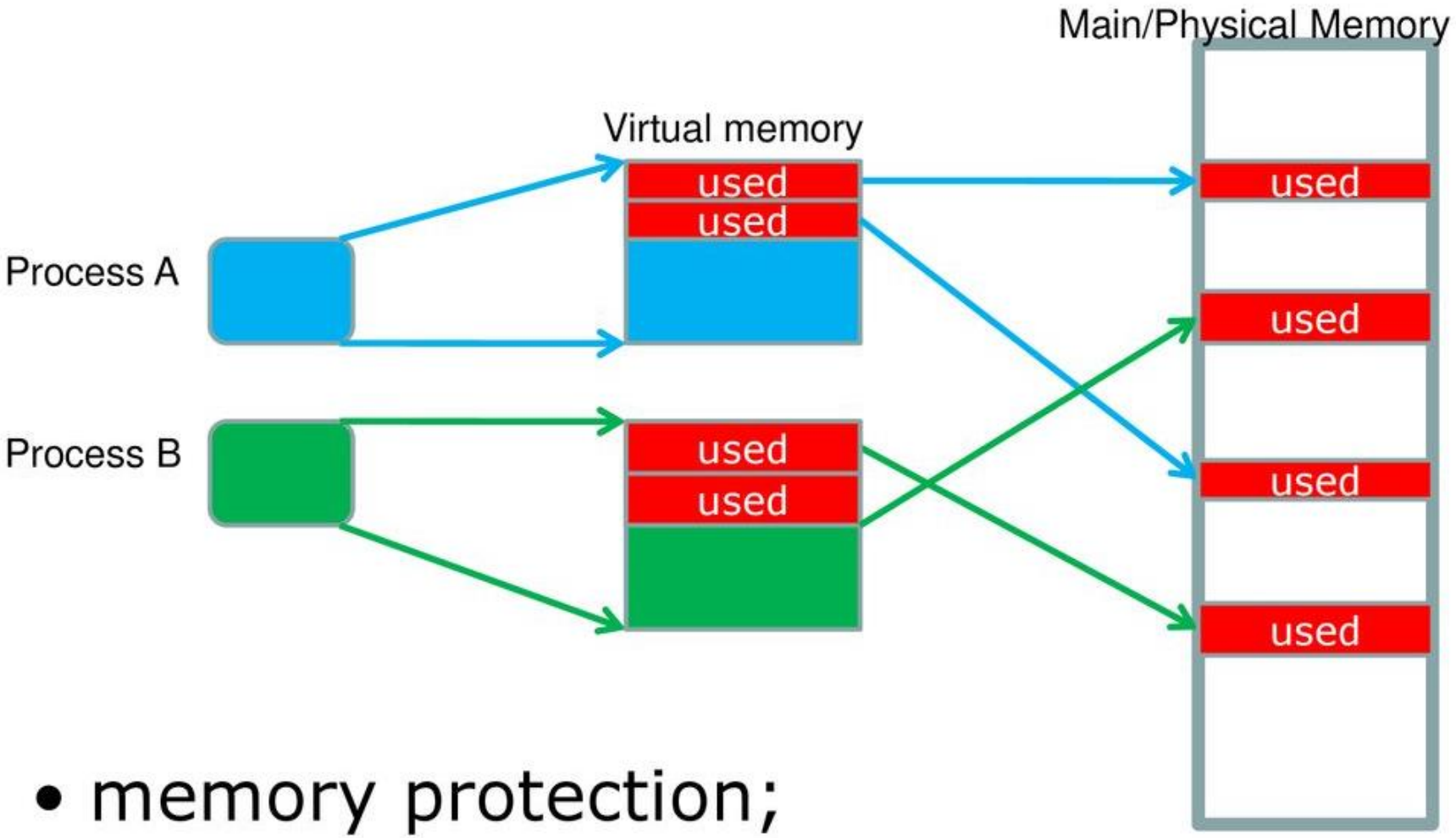
Các kiểu lưu trữ thông tin trên máy tính



RAM

- RAM (random access memory)
  - Một dãy các byte liên tiếp (một mảng byte khổng lồ)
  - Mọi thứ đều nằm trên đó
    - Hệ điều hành
    - Các trình điều khiển thiết bị
    - Các chương trình
      - Mã chương trình
      - Hằng số, trực trị
      - Biến
      - ...
- Do tất cả đều nằm trên bộ nhớ, về lý thuyết:
  - Có thể biết chính xác "địa chỉ" của chúng?
  - Có thể "tóm" được chúng và đọc / ghi giá trị?

Bộ nhớ vật lý và bộ nhớ bảo vệ



- memory protection; process isolation

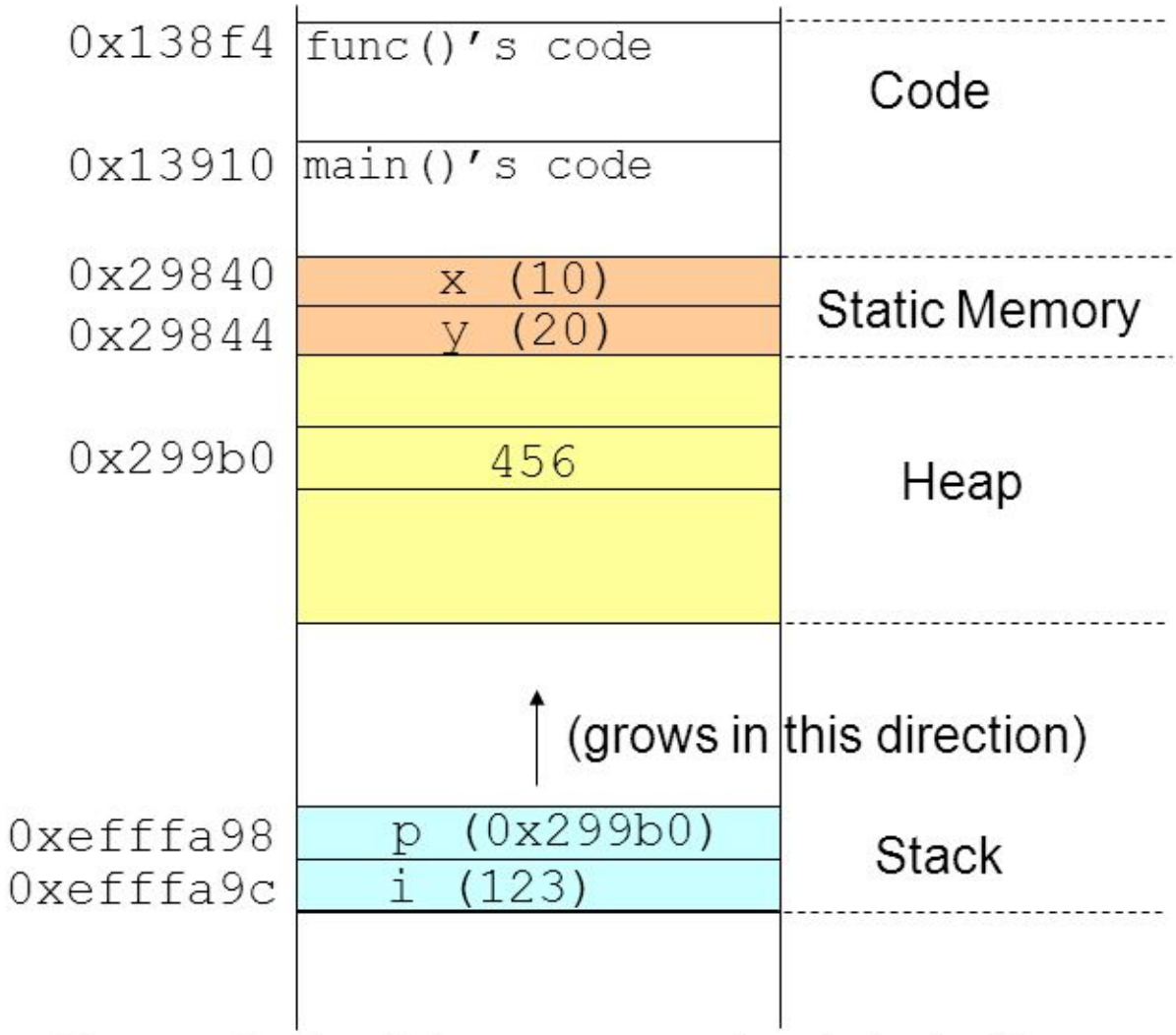
Bộ nhớ của chương trình C/C++

```
int x = 10
int y = 20;

void func() {
   ...
}

int main()
{
    int i = 123;
    int *p;

    p = new int;
    *p = 456;
    ...
}
```



Phần 2

Biến và địa chỉ của biến

Biến và địa chỉ của biến

- Biến nằm trong bộ nhớ, nó phải nằm ở một (vài) ô nhớ nào đó, vị trí này gọi là địa chỉ (address) của biến
- Phép toán địa chỉ: &
  - Trả về địa chỉ của biến
  - Thường là một số 32 bit (tùy vào CPU, OS và kiểu chương trình)
  - In ra màn hình ở dạng hexadecima
- Ví dụ:

```
int a[] = { 1, 3, 2, 4, 2 };
cout << &a << endl;
cout << &a[0] << endl;
cout << &a[1] << endl;
cout << (long) &a[2] << endl;
```

- Có lấy được địa chỉ của thứ khác trong bộ nhớ không?

Phần 3

Biến con trỏ

Biến con trỏ

- Con trỏ = Kết quả của phép lấy địa chỉ &
- Ta có thể lưu trữ kết quả này vào một biến hay không?
  - Có, sử dụng biến có kiểu "con trỏ"
  - Khai báo như biến bình thường, thêm dấu * trước tên biến
- Ví dụ:

```
int a = 10;
int *pa = &a;     // con trỏ tới biến a

cout << "A = " << a << endl;
cout << "PA (con tro) = " << pa << endl;
cout << "PA (int) = " << (int) pa << endl;
```

- Thậm chí có thể khai báo lẫn lộn:

```
int a, *b, c, **d;
```

Khai báo và khởi tạo con trỏ

- Khai báo: như một biến thông thường, hơi rối hơn chút

```
int *p1;          // con trỏ đến giá trị int
double *p2;       // con trỏ đến giá trị thực
bool *p3;         // con trỏ đến giá trị logic
int **p4;         // con trỏ đến con trỏ kiểu nguyên
```

- Khởi tạo:

```
int n;
int *p1 = &n;         // con trỏ đến n
double *p2;           // con trỏ đến đâu???
bool *p3 = NULL;      // con trỏ NULL
```

- Nếu không khởi tạo thì sao? Có thể gây lỗi
- Khởi tạo có chắc chắn an toàn? Không chắc
  - `NULL` là một giá trị đặc biệt, bằng 0 (`nullptr` từ C++11)

Sử dụng con trỏ

- Con trỏ thì có ích gì?
  - Máy tính dùng con trỏ trong thao tác bộ nhớ, đoạn mã dùng con trỏ sẽ có tốc độ cao hơn do dễ dàng dịch thành các mã máy tương ứng (lý do ngôn ngữ lập trình C/C++ chạy nhanh)
  - Biết địa chỉ của biến, biết biến đó nằm ở đâu trong bộ nhớ
  - Thông qua con trỏ, có thể truy cập vào biến để đọc/ghi giá trị
- Phép toán truy cập với con trỏ: *
- Ví dụ:

```
int n = 1, m = 2;
int *p = &n;          // p trỏ đến n
*p = 100;             // n = 100
p = &m;               // p trỏ đến m
*p = 200;             // m = 200
```

Con trỏ làm tham số của hàm: có gì đặc biệt?

```
#include <iostream>

using namespace std;

void change(int * a) {
    *a = 10;
    cout << "a = " << *a << endl;
}

int main() {
    int b = 5;
    cout << "b = " << b << endl;
    change(&b);
    cout << "b = " << b << endl;
}
```

① In ra b (b = 5)

② Phát lời gọi hàm, gán tham số: a = &b (*a ≈ b)

③ Thay đổi *a = 10 (b = 10)

④ In ra *a (*a = 10)

⑤ Thoát khỏi hàm

⑥ In ra b (b = 10)

Muốn tham số bị thay đổi giá trị trong hàm: sử dụng con trỏ

Quy tắc sử dụng con trỏ

- Con trỏ là khái niệm quan trọng và khó trong C/C++
  - Nhiều công ty phần mềm đánh giá mức độ thành thạo C/C++ qua khả năng hiểu và sử dụng con trỏ của ứng viên
- Hai phép toán đối lập: & và *
  - Phép & trả về địa chỉ của biến
  - Phép * trả về biến từ địa chỉ
- Quy tắc cặp đôi: `int a, *pa = &a;`
  - *pa và a đều chỉ nội dung của biến a
    - *pa còn được gọi là truy cập gián tiếp vào a
  - pa và &a đều là địa chỉ của biến a
- Con trỏ phải được khởi tạo, nếu chưa biết trỏ đến biến nào thì gán bằng `NULL` / `nullptr`

Phép toán trên con trỏ

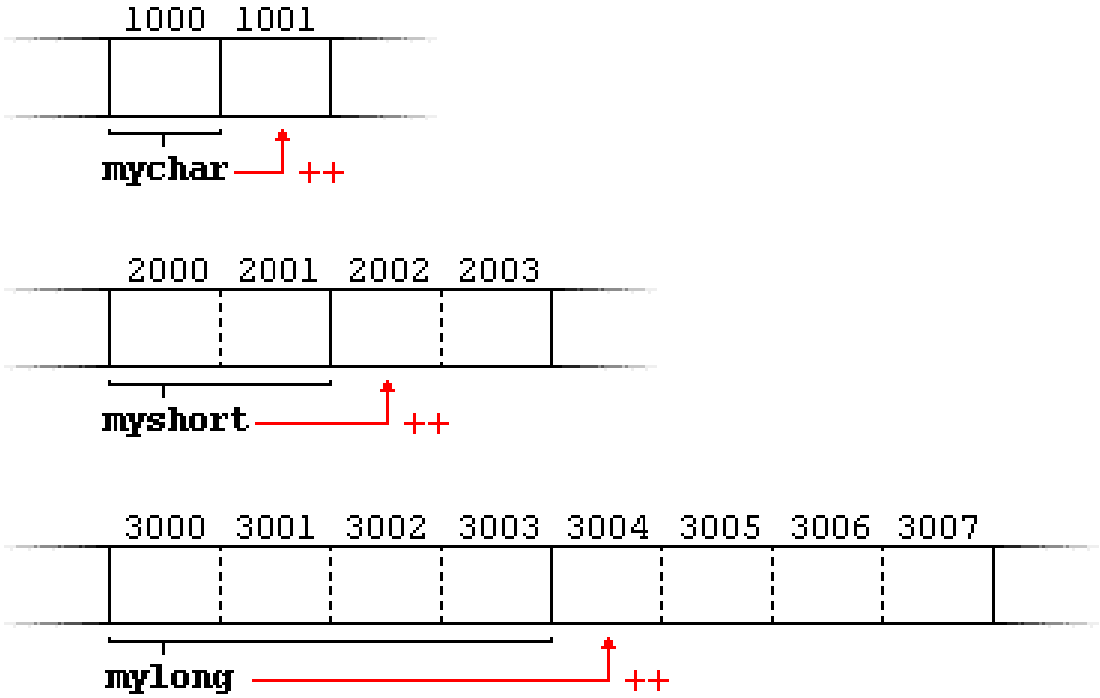
- Phép lấy biến (đã học): *pa
- Phép gán: p1 = p2
  - Hai con trỏ bằng nhau, trỏ đến cùng một chỗ

```
p1 = p2;
```


```
*p1 = *p2;
```

Phép toán trên con trỏ

- Phép cộng với số nguyên: pa + n
  - Tăng / Giảm địa chỉ, trả về con trỏ đến biến tiếp theo trong bộ nhớ (chú ý!)
  - Giá trị n có thể âm
  - Có thể dùng ++ , += hoặc --, -=



Phép toán trên con trỏ

- Tính khoảng cách giữa hai con trỏ: pa-pb
  - Đối ngẫu với phép cộng con trỏ với số nguyên
  - Kết quả trả về là số nguyên
  - Là khoảng cách giữa hai con trỏ, tính bằng số lượng biến cùng kiểu với con trỏ
  - Rất cẩn thận khi sử dụng
- Chú ý ví dụ sau:

```
int a; int b;
int *pa = &a, *pb = &b;
cout << pb-pa << endl;
short *ppa = (short *) pa;
short *ppb = (short *) pb;
cout << ppb-ppa << endl;
```

Phép toán trên con trỏ

- Giữa các con trỏ có thể sử dụng các phép so sánh thông thường, vì bản chất chúng là các giá trị số
  - So sánh địa chỉ 2 ô nhớ
  - Thực hiện được cả 6 phép so sánh
    - ==
    - !=
    - >
    - >=
    - <
    - <=
  - Không có nhiều ý nghĩa, ngoại trừ phép == và phép !=

Phần 4

Mảng và con trỏ

Mảng và con trỏ

- Mảng (một chiều) và con trỏ trong C/C++ có một số đặc điểm giống nhau về cách sử dụng, thậm chí sử dụng có phần lẫn lộn
  - Vì lý do đó nên một số tài liệu xem mảng là hằng con trỏ (tức là một con trỏ nhưng trỏ đến một vị trí cố định trong bộ nhớ), điều này không hoàn toàn chính xác
  - Cách tốt nhất là hãy phân biệt rạch ròi giữa mảng và con trỏ, cho dù chúng có nhiều đặc điểm chung
- Mảng có thể gán cho con trỏ, có vẻ như chúng là một

```
int a[5], *p1, *p2;
p1 = a;              // p trỏ đến đầu của a, tức a[0]
p2 = &a[0];          // p trỏ đến a[0]
cout << p1 << endl; // in ra địa chỉ của a
cout << p2 << endl; // in ra địa chỉ của a[0], giống p1
```

Mảng và con trỏ

- Mặc dù có thể dùng biến mảng như biến con trỏ

```
int a[5] = { 1, 2, 3, 4, 5 }, *p = a;
cout << a[2] << endl;        // 3, bình thường
cout << *(a+2) << endl;      // 3, dùng a như con trỏ
cout << *(p+2) << endl;      // 3, bình thường
cout << p[2] << endl;        // 3, dùng p như mảng
cout << *(2+p) << endl;      // 3, lạ chưa?
```

- Nhưng có vẻ C/C++ không xem hai biến là giống nhau

```
int a[5], *p = a;
cout << sizeof(a) << endl;  // 20
cout << sizeof(p) << endl;  // 4
```

- Dưới đây là cái gì?

```
int *a[5];
int **p;
```

Mảng và con trỏ

- Thậm chí mảng trong C/C++ đôi khi cũng không giống chính nó:
  - Mảng trong hàm main là hằng số
  - Nhưng mảng a là tham số của hàm print thì không
  - Bạn có lời giải thích nào không?

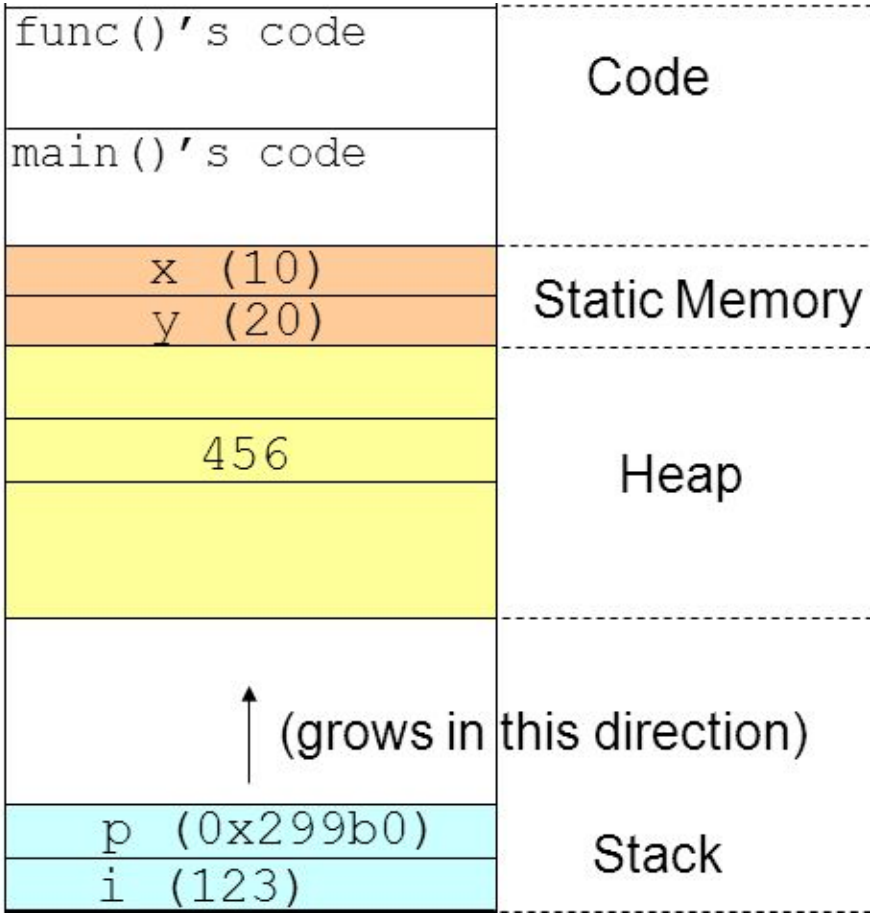
```
void print(int a[], int n) {
    for (int i = 0; i < n; i++)
        cout << *(a++) << " ";
}
int main() {
    int n = 9, a[] = { 6, 5, 9, -1, 100, 7, 5, 5, 2 };
    print(a, n);
    for (int i = 0; i < n; i++)
        cout << *(a++) << " ";       // lỗi
}
```

Phần 5

Bộ nhớ động

Bộ nhớ động

- Cơ chế bộ nhớ:
  - Vùng code: chứa mã thực thi
  - Vùng data (static memory): chứa các dữ liệu được khởi tạo từ ban đầu, thường là các biến global
  - Vùng stack: chứa các biến địa phương
    - Vùng này sẽ tăng giảm theo độ sâu gọi hàm (call stack)
  - Vùng heap: chứa các biến sẽ được "cấp phát động"
    - Chẳng hạn như dữ liệu của vector, có thể lúc ít phần tử, lúc khác lại chứa rất nhiều phần tử



Bộ nhớ động

- Cấp phát các biến trong vùng nhớ heap gọi là "cấp phát động" – khi nào cần mới yêu cầu cấp
  - Hãy tưởng tưởng phần mềm Microsoft Word cần những biến nào để hiển thị và soạn thảo một file?
  - Không thể biết trước được, có những file rất ít dữ liệu, có những file cực nhiều dữ liệu
  - Trong trường hợp này phù hợp nhất là dùng cơ chế cấp phát động, xin bộ nhớ theo nhu cầu của phần mềm
- Có 2 cơ chế cấp phát động thông dụng
  - Cấp phát theo khối nhớ, sử dụng các hàm cũ của C
  - Cấp phát theo đối tượng, dùng cơ chế tạo đối tượng của C++
  - Một số tài liệu nói C cấp phát ở "heap" còn C++ cấp phát ở "free store", thực chất hai vùng nhớ này là một

Cấp phát động kiểu C: cấp theo khối nhớ

- Sử dụng thư viện: <stdlib.h>
- Bốn hàm chính:
  - malloc(N) – cấp một khối nhớ cỡ N byte
  - calloc(N, S) – cấp một khối nhớ cỡ N x S byte, điền số 0 vào mọi ô dữ liệu được cấp phát
  - free(p) – hủy khối nhớ được cấp cho con trỏ p
  - realloc(p, S) – chỉnh kích cỡ khối nhớ được cấp bởi con trỏ p thành cỡ S byte, giữ lại dữ liệu cũ đã được khởi tạo
- Nguyên tắc:
  - Các hàm cấp phát trả về con trỏ void (void *)
  - Cần chuyển kiểu để dễ sử dụng
  - Cấp phát không thành công sẽ trả về con trỏ nullptr
  - Cấp phát xong, không dùng thì nên hủy

Cấp phát động kiểu C: cấp theo khối nhớ

```
#include <iostream>
#include <stdlib.h>
using namespace std;

const int N = 100;

int main() {
    double *ptr;
    ptr = (double *) malloc(N * sizeof(double));
    if (ptr == NULL) cout << "Lỗi cấp phát động";
    else
        // in ra xem dữ liệu được khởi tạo thế nào
        for (int i = 0; i < N; ++i)
            cout << *(ptr++) << endl;
    // thực ra không cần lắm
    free(ptr);
}
```

Cấp phát động kiểu C++: cơ chế tạo đối tượng

- Không cần thư viện
- Sử dụng các phép toán thay vì lệnh
  - Toán tử new: tạo đối tượng (biến)
  - Toán tử new[]: tạo mảng đối tượng
  - Toán tử delete: hủy đối tượng
  - Toán tử delete[]: hủy mảng đối tượng
- Chú ý:
  - Cấp phát xong không dùng thì nên hủy
  - Cấp phát mảng thì phải dùng hủy mảng
  - Cấp phát kiểu C++ chậm hơn so với kiểu của C vì toán tử new sau khi cấp phát bộ nhớ thì luôn luôn khởi tạo giá trị mặc định cho biến

Cấp phát động kiểu C++: cơ chế tạo đối tượng

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

const int N = 5;

int main() {
    double *ptr = new double [N];
    if (ptr == NULL) cout << "Lỗi cấp phát động";
    else
        // in ra xem dữ liệu được khởi tạo thế nào
        for (int i = 0; i < N; ++i)
            cout << *(ptr++) << endl;
    // thực ra không cần lắm
    delete [] ptr;
}
```

Phần 6

Con trỏ hàm

Con trỏ hàm

- Câu hỏi ở slide 9: ngoài việc lấy địa chỉ của biến, còn có thể lấy được địa chỉ của "các thứ khác" không?
  - Và nếu lấy được thì dùng vào việc gì?
- C/C++ cho phép ta lấy địa chỉ của một hàm, và lưu nó vào biến con trỏ, biến này gọi là "con trỏ hàm"
- Có thể gọi hàm thông qua con trỏ hàm
- Khai báo: <kiểu trả về> (* <biến>)(<các tham số>);

```
// haingoi: con trỏ đến một hàm có 2 tham số kiểu
// nguyên và trả về kiểu nguyên
int (*haingoi) (int a, int b);
// inketqua: con trỏ đến một hàm có tham số kiểu mảng
// và số nguyên n, hàm không trả về giá trị
void (*inketqua) (int a[], int n);
```

Con trỏ hàm: gán qua tên hàm

- Sau khi khai báo xong, ta dùng con trỏ hàm như một biến con trỏ thông thường, nhưng nhận kết quả là địa chỉ một hàm, hàm này phải khớp với khai báo ở biến

```
// khai báo hàm
int foo();
double goo();
int hoo(int x);
// gán thử con trỏ hàm
int (*p1)() = foo;          // ok
int (*p2)() = goo;          // lỗi, sai kiểu trả về
double (*p3)() = &goo;      // ok, viết thế cũng được
p1 = hoo;                   // lỗi, sai tham số
int (*p4)(int) = hoo;       // ok
p2 = nullptr;               // ok
```

Con trỏ hàm: gọi hàm qua tên biến

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

double tong(double a, double b) {
    return a + b;
}

double tich(double a, double b) {
    return a * b;
}

int main() {
    double (*p) (double, double) = tong;
    cout << "Ket qua 1 = " << p(10, 20) << endl;
    p = tich;
    cout << "Ket qua 2 = " << (*p)(10, 20) << endl;
}
```

Con trỏ hàm: dùng làm tham số của hàm khác

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

int tong(int a, int b) {
    return a + b;
}
int tich(int a, int b) {
    return a * b;
}
void inketqua(int a, int b, int (*p) (int, int)) {
    cout << "Ket qua = " << p(a, b) << endl;
}
int main() {
    inketqua(10, 20, tong);
    inketqua(10, 20, tich);
}
```

Con trỏ hàm: hàm trả về con trỏ hàm

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

int tong(int a, int b) { return a + b; }
int tich(int a, int b) { return a * b; }

int (*chonham(int n)) (int, int) {
    if (n == 0) return tong;
    else return tich;
}
void inketqua(int a, int b, int (*p) (int, int)) {
    cout << "Ket qua = " << p(a, b) << endl;
}
int main() {
    inketqua(10, 20, chonham(1));
    inketqua(10, 20, chonham(0));
}
```

Phần 7

Bài tập

Bài tập

1. Cho đoạn mã sau

```
double d, *p;
d = 123.45;
p = &d;
```

hãy cho biết giá trị của:

a. d
b. &d
c. *d
d. *&d
e. &*d
f. p
g. &p
h. *p
i. *&p
j. &*p

Bài tập

2. Cho đoạn mã sau

```
int  *pi;
float f;
char c;
double *pd;
```

Những lệnh nào dưới đây có lỗi? Tại sao?

a. `f = *pi;`
b. `c = *pd;`
c. `*pi = *pd;`
d. `*pd = f;`
e. `pd = f;`
f. `*pd = *pi;`
g. `*pi = c;`
h. `*pi = pd;`

Bài tập

3. Sử dụng con trỏ, viết các hàm sau:
   a. Hàm doicho, tráo đổi giá trị của hai số thực cho nhau.
   b. Hàm khoitao, có nhiệm vụ gán giá trị bằng 0 cho n số thực liên tiếp trong bộ nhớ. Hàm nhận tham số thứ nhất là con trỏ đến số thực đầu tiên và n là tham số thứ hai.
   c. Hàm saochep, có nhiệm vụ sao chép n số nguyên liên tiếp từ vùng nhớ pa sang vùng nhớ pb. Hàm nhận tham số thứ nhất là số n, con trỏ pa đến đầu khối dữ liệu nguồn, con trỏ pb đến đầu khối dữ liệu đích.
   d. Hàm nhonhat, nhận tham số là mảng A nguyên và số phần tử n, trả về địa chỉ của phần tử nhỏ nhất. Trường hợp có nhiều phần tử nhỏ nhất thì trả về địa chỉ của phần tử cuối cùng. Trường hợp không có phần tử nhỏ nhất thì trả về NULL (hoặc nullptr).

Bài tập

4. Sử dụng con trỏ, viết các hàm sau:
   a. Hàm sotiep, nhận tham số là một con trỏ nguyên pa và số nguyên k. Hàm bắt đầu tìm từ pa+1 tăng dần cho đến khi thấy số k thì trả về con trỏ đến số đó.
   b. Hàm sotiep2, nhận tham số là một con trỏ nguyên pa, số nguyên k và con trỏ nguyên px. Hàm bắt đầu tìm từ pa+1 tăng dần cho đến khi thấy số k hoặc chạm đến px. Nếu tìm thấy số k thì trả về con trỏ đến số đó, nếu chạm đến px (và không tìm thấy số k) thì trả về nullptr.
   c. Hàm nghichdao, nhận tham số thứ nhất là số nguyên n, hai tham số tiếp theo là hai con trỏ thực pa và pb. Hàm copy n số thực từ pa sang pb nhưng đảo ngược lại thứ tự (tức là dãy pa được copy nội dung sang dãy pb nhưng thứ tự ngược lại)

Bài tập

5. Viết hàm khoitao(n) trả về một khối n số nguyên (con trỏ nguyên) được cấp phát động và khởi tạo sẵn giá trị từ 1 đến n.
6. Viết hàm danhsach(n) trả về một mảng n chuỗi, tất cả đều được khởi tạo nội dung là "DHTL Ha Noi"
7. Viết hàm saochep(a, n), hàm nhận 2 tham số mảng số thực a có n phần tử, hàm sẽ trả về bản sao của mảng a ở dạng con trỏ, bản sao này có dữ liệu giống hệt mảng a.
8. Viết hàm doancon(a, d, c), hàm nhận tham số mảng bool a, hàm tạo ra bản sao đoạn con của a từ vị trí d đến vị trí c, sau đó trả về bản sao này ở dạng con trỏ.

Bài tập

9. Viết hàm xoahet(a, n, b) trong đó a là một mảng số nguyên có n phần tử, hàm sẽ tìm các phần tử có giá trị bằng b trong a và xóa hết chúng đi, sau đó hàm trả về số lượng phần tử còn lại trong a.
10. Viết hàm khoitao(a, b, c) trong đó a, b, c là 3 con trỏ kiểu double; hàm làm nhiệm vụ cấp phát động 3 số thực riêng rẽ và lần lượt gán chúng vào các con trỏ a, b, c để trả về hàm gọi.
11. Viết hàm ghepnoi(a, n) trong đó a là một mảng n string, hàm sẽ chuyển đổi a thành một string bằng cách ghép liên tiếp nội dung của các string thuộc a với nhau, hàm trả về con trỏ là kết quả thực hiện được.