Tiến bộ con trỏ

Mục tiêu

-Con trỏ cấp 2

-Con trỏ & mảng đa chiều

-Mảng con trỏ

-Con trỏ hàm

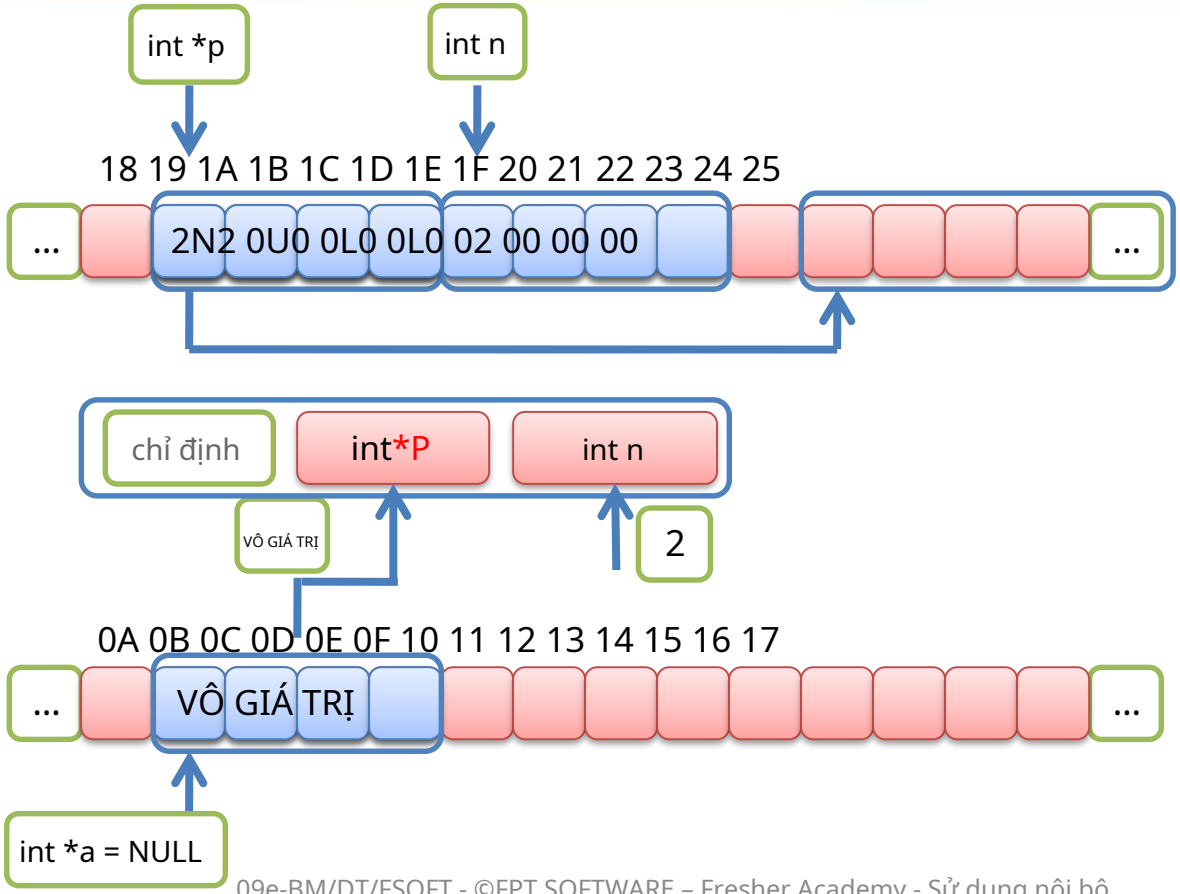
Con trỏ cấp 2 (con trỏ tới con trỏ)

- Vấn đề

Làm cách nào để thay đổi giá trị của con trỏ (không phải giá trị mà nó trỏ tới) sau khi gọi hàm?

```
void Allocate(int *p, int n) {

        p = (int *)malloc(n * sizeof(int));
}
khoảng trống chính()
{
        int *a = NULL;
        Phân bổ (a, 2);
        // a tĩnh = NULL
}
```

Con trỏ cấp 2 - 1



Con trỏ cấp 2 - 2

- Giải pháp
  - Sử dụng tài liệu tham khảoint *&p(trong C++)

```
void CapPhat(int *&p, int n) {

        p = (int *)malloc(n * sizeof(int));
}
```

  - Không thay đổi tham số trực tiếp và trả về

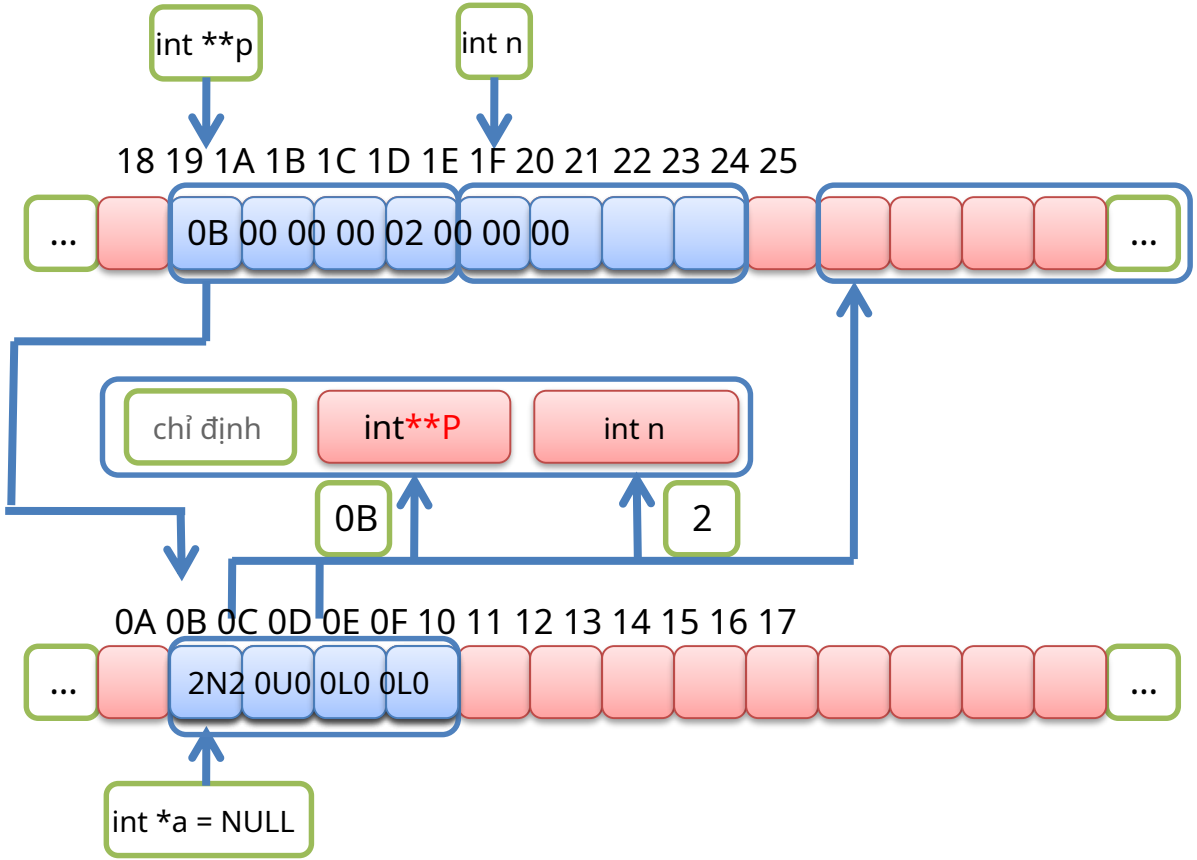
```
int*Phân bổ(int n)
{
        int *p = (int *)malloc(n * sizeof(int)); trả lại p;

}
```

Con trỏ cấp 2 - 3

- Giải pháp
  - Dùng con trỏ p trỏ tới con trỏ a. Hàm sẽ thay đổi giá trị của con trỏ a một cách gián tiếp thông qua con trỏ p.

```
khoảng trống Phân bổ(int **p, int n) {

        * p = (int *)malloc(n * sizeof(int));
}

khoảng trống chính()
{
        int *a = NULL;
        Phân bổ (&a, 4);
}
```

Con trỏ cấp 2 - 4



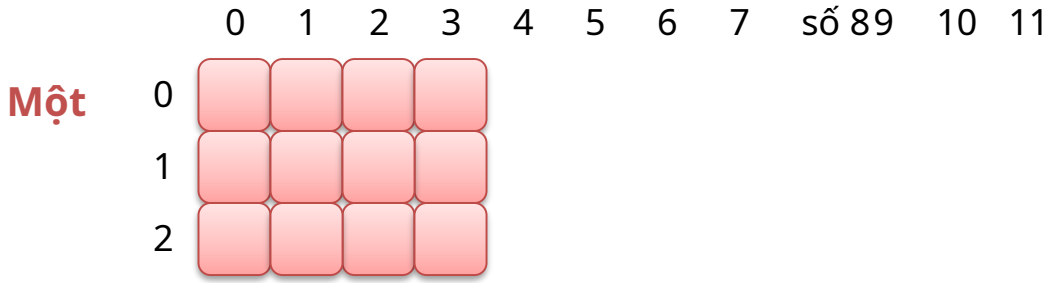
Con trỏ cấp 2 - 5

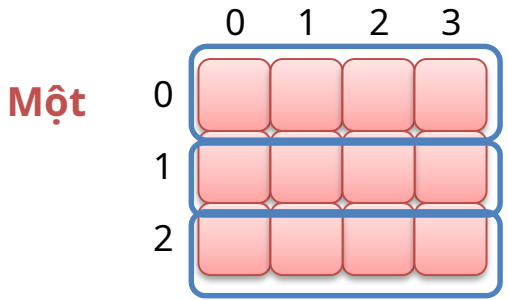
- Ghi chú

```
int x = 12;
int *ptr = &x;                          // ĐƯỢC RỒI
int k = &x; ptr = k;                    // Lỗi

int **ptr_to_ptr = &ptr; int            // ĐƯỢC RỒI
**ptr_to_ptr = &x;                      // Lỗi

* * ptr_to_ptr = 12;                    // ĐƯỢC RỒI
* ptr_to_ptr = 12;                     // Lỗi

printf("%d", ptr_to_ptr); printf("%d", // Địa chỉ của ptr
printf("%d", **ptr_to_ptr); // Giá trị của x  // Giá trị của ptr
```

Con trỏ & mảng 2 chiều - 1

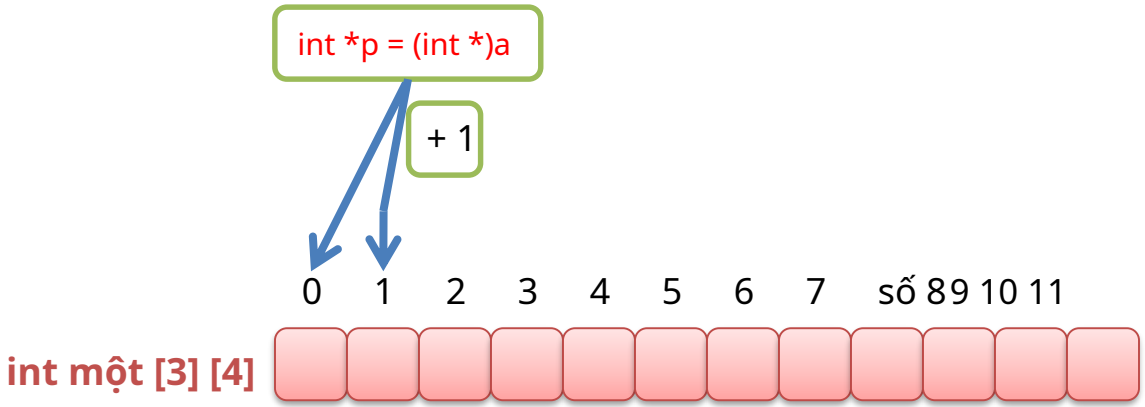
int a[3][4];





Con trỏ & mảng 2 chiều - 2

- Phương pháp 1
  - Các phần tử tạo mảng 1 chiều
  - Sử dụng con trỏ int * để truy cập mảng 1 chiều



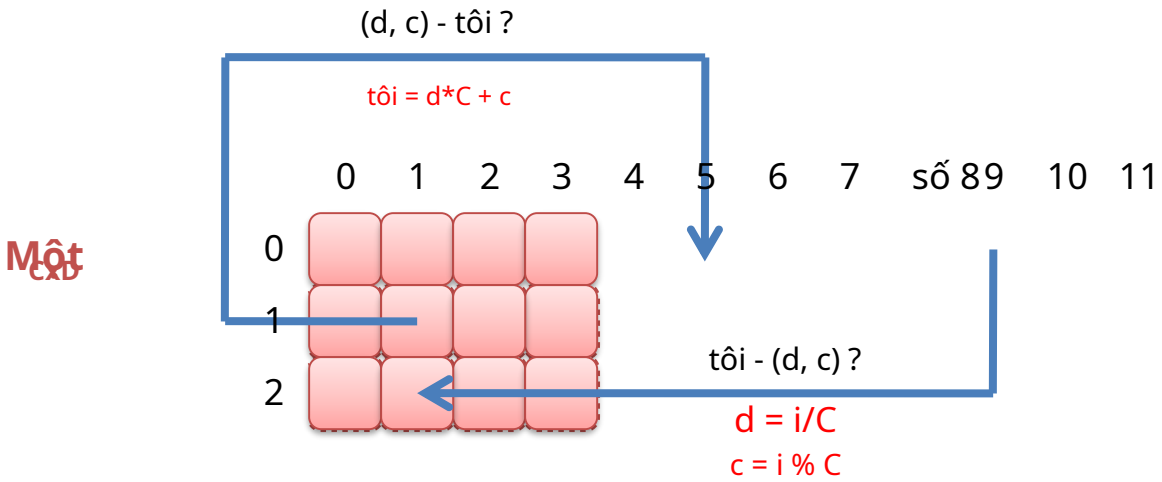
Cách 1 - 1

- Nhập/xuất theo chỉ số của mảng 1 chiều

```
# định nghĩa   D  3
# định nghĩa   C  4
khoảng trống chính()
{
        int a[D][C], i;
        int *p =(int *)Một;
        cho (i = 0; i < D*C; i++) {

                printf(“Phần tử đầu vào %d: ”, i);
                scanf(“%d”, p + i);
        }

        cho (i = 0; i < D*C; i++)
                printf(“%d ”, *(p + i));
}
```

Cách 1 - 2

- Mối liên hệ giữa chỉ số của mảng 1 và 2 chiều



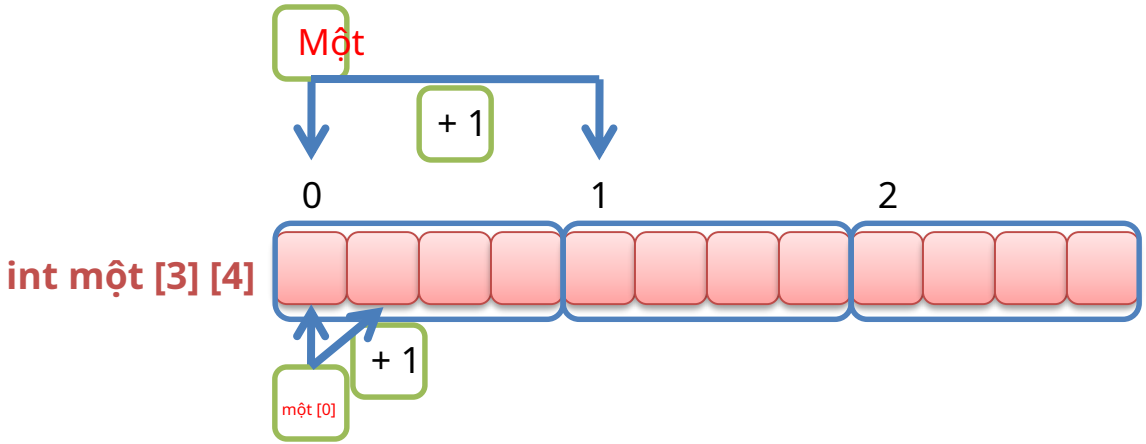
Cách 1 - 3

- Nhập/xuất theo chỉ số của mảng 2 chiều

```
int a[D][C], i, d, c; int *p =
(int *)a;

cho (i = 0; i < D*C; i++) {

        printf(“Nhập a[%d][%d]: ”, i/C, i % C); scanf(“%d”, p + i);

}
vì (d = 0; d < D; d++) {

        vì (c = 0; c < C; c++)
                printf(“%d ”, *(p + d * C + c)); // *p++
                printf(“\n”;
}
```

Con trỏ và mảng 2 chiều

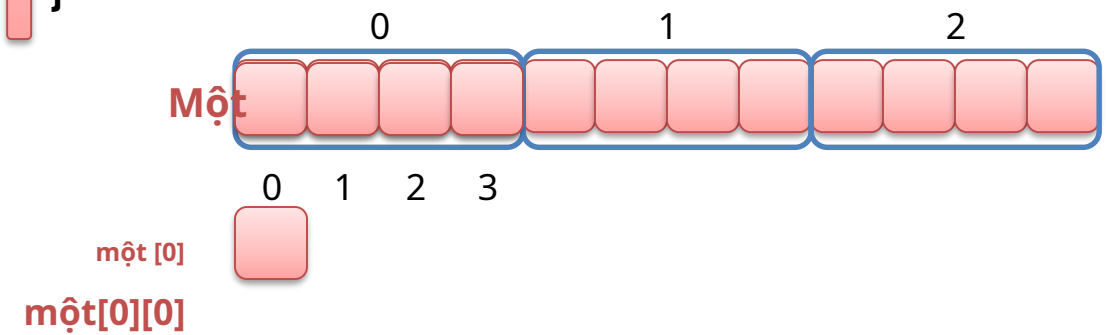
- Phương pháp 2
  - Mảng 2 chiều, mỗi phần tử là một mảng 1 chiều
    - a chứa a[0], a[1], ... - a = &a[0]
    - a[0] chứa a[0][0], a[0][1], ... - a[0] = &a[0][0]



Cách 2 - 1

- Kích thước của mảng

```
khoảng trống chính()
{
        int a[3][4];
        printf("Kích thước của a = %d", sizeof(Một)); printf("Kích
        thước của a[0] = %d", sizeof(một [0])); printf("Kích thước của
        a[0][0] = %d", sizeof(một[0][0]));
}
```



Cách 2 - 2

- Bình luận
  - a trỏ tới a[0], a[0] trỏ tới a[0][0] - a là con trỏ cấp 2.
  - Truy cập a[0][0] bằng 3 cách:

```
khoảng trống chính()
{
        int a[3][4];
        a[0][0] = 1;
        * a[0] = 1;
        * * a = 1;

        a[1][0] = 1; *a[1] = 1; **(a+1) = 1;
        Một[1][2] = 1; *(Một[1]+2) = 1; *(*(a+1)+2) = 1;
}
```

Cách 2 - 3

- Truyền mảng vào hàm
  - Vượt quađịa chỉ của phần tử đầu tiênđể hoạt động.
  - Khai báo con trỏ và gán địa chỉ mảng cho con trỏvì vậy nó trỏ đến mảng.
  - Con trỏ phải cùng kiểu với mảng, nghĩa là con trỏ trỏ tới của n phần tử. ký ức

- Cú pháp

- Ví dụ

<kiểu dữ liệu> (*<tên con trỏ>)[<số phần tử>];

int (*ptr)[4];

Cách 2 - 4

- Truyền mảng vào hàm

```
void Output_1_Array_C1(int (*ptr)[4])                    // ptr[][4]
{
        int *p =(int *)ptr; vì (int i = 0; i < 4;
        i++)
                printf("%d ", *p++);
}
trống rỗng  chủ yếu()
{
        int a[3][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12}};
        int (*ptr)[4];
        ptr = a;
        vì (int i = 0; i < 3; i++)
                Output_1_Array_C1(ptr++); //hoặc ptr + tôi Đầu
                ra_1_Array_C1(một++); //sai => a + tôi
}
```

Cách 2 - 5

- Truyền mảng vào hàm

```
void Output_1_Array_C2(int *ptr, int n) // ptr[] {

        cho (int i = 0; i < n; i++)
                printf("%d ", *ptr++);
}
trống rỗng   chủ yếu()
{
        int a[3][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12}};
        int (*ptr)[4];
        ptr = a;
        vì (int i = 0; i < 3; i++)
                Đầu ra_1_Array_C2((int *)ptr++); Đầu ra_1_Array_C2(
                (int *)(a + i));//một++
sai
}
```

Cách 2 - 6

- Truyền mảng vào hàm

```
void Output_n_Array_C1(int (*ptr)[4], int n) {

        int *p = (int *)ptr;
        vì (int i = 0; tôi <n*4; tôi++)
                printf("%d ", *p++);
}
trống rỗng   chủ yếu()
{
        int a[3][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12}};
        int (*ptr)[4];
        ptr = a;

        Output_n_Array_1(ptr, 3);
        Output_n_Array_1(a, 3);
}
```

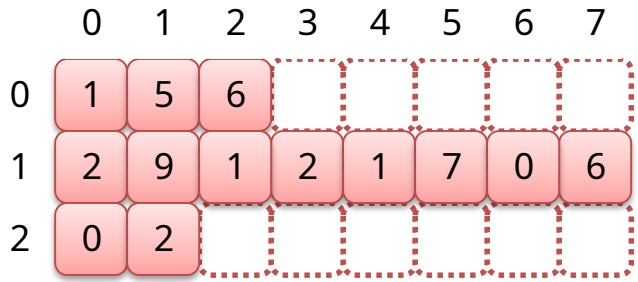
Cách 2 - 7

- Truyền mảng vào hàm

```
void Output_n_Array_C2(int (*ptr)[4], int n)
{
        int *p;
        cho (int i = 0; i < n; i++)
        {
                p = (int *)ptr++;

                vì (int i = 0; i < 4; i++)
                        printf("%d ", *p++);
                printf("\n");
        }
}
```

Mảng con trỏ - 1

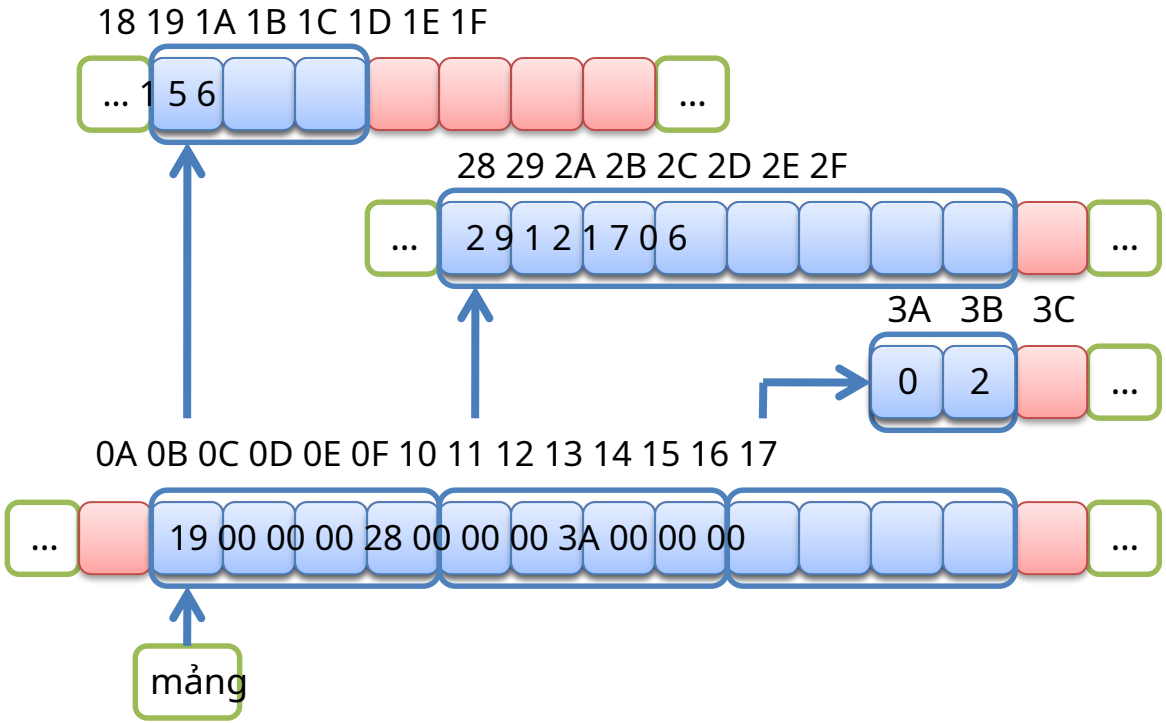
- Vấn đề
  - Sử dụng cấu trúc dữ liệu nào để lưu trữ dữ liệu dưới đây?



- Giải pháp?
  - Cách 1: Mảng 2 chiều 3x8 (bộ nhớ lãng phí)

Mảng con trỏ - 2

- Cách 2: Mảng con trỏ 1 chiều



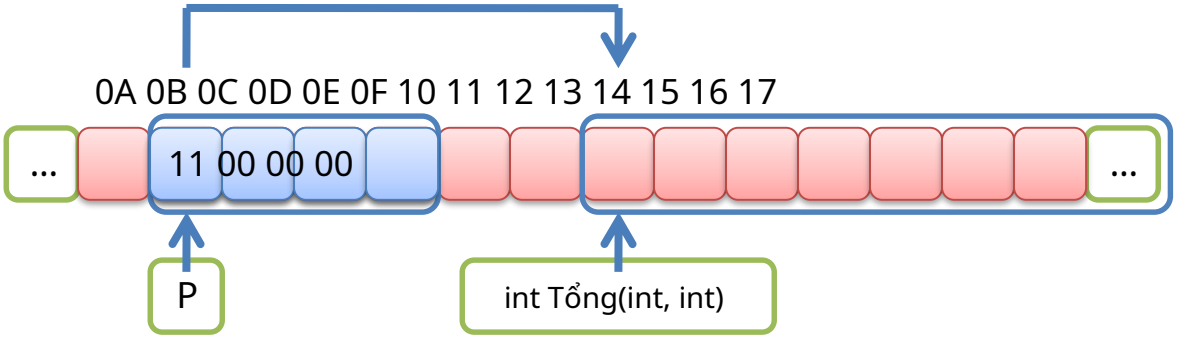
Mảng con trỏ - 3

- Ví dụ

```
void print_strings(ký tự *p[], int n)
{
        cho (int i = 0; i<n; i++)
                printf("%s ", p[i]);
}

khoảng trống chính()
{
        char *message[4] = {"Fpt", "Phần mềm", "Lực
lượng lao động", "Đảm bảo"};

        print_strings(tin nhắn, 4);
}
```

Con trỏ hàm - 1

- Ý tưởng
  - Chức năngđược lưu trữ trong bộ nhớ, chúng cóĐịa chỉ.
  - Con trỏ hàm là con trỏ trỏ tới bộ nhớ của hàm và gọi hàm thông qua con trỏ.



Con trỏ hàm - 2

- Tuyên bố độc quyền

```
<kiểu trả về>(*<tên con trỏ>)(danh sách tham số);
```

- Ví dụ

```
// Con trỏ hàm với tham số int, trả về int
int (*ptof1)(int x);

// Con trỏ hàm có 2 tham số double, không trả về gì
void (*ptof2)(gấp đôi x, gấp đôi y);

// Con trỏ hàm với tham số mảng, trả về char
char (*ptof3)(char *p[]);

// Con trỏ tới hàm không có đối số và không trả về gì
void (*ptof4)();
```

Con trỏ hàm - 3

- Khai báo ngầm định (thông qua loại)

```
typedef <kiểu trả về> (* <tên kiểu>)(danh sách thông số); <gõ nam> <tên con trỏ>;
```

- Ví dụ

```
int (*pt1)(int, int);                    // rõ ràng

typedef int (*Toán tử)(int, int);

Toán tử pt2, pt3; // ngầm định
```

Con trỏ hàm - 4

- Gán giá trị cho con trỏ hàm

```
<con trỏ func> = <tên func>; <func
poiter> = &<tên func>;
```

  - Hàm được gán phải có cùng nguyên mẫu (đầu vào, đầu ra)
- Ví dụ

```
int Sum(int x, int y);                          // Chức năng
int Phép trừ(int x, int y);                     // Chức năng
int (*tính toán)(int x, int y); // con trỏ hàm

tính toán     =  Tổng;        // kiểu ngắn
tính toán     =  &Phép trừ;           // sử dụng địa chỉ
tính toán = NULL;             // trỏ đến không có gì
```

Con trỏ hàm - 5

- So sánh con trỏ hàm

```
nếu như (tính toán != NULL)
{
        nếu như (tính == &Tổng
                printf("Hàm con trỏ tính tổng");
        khác
                nếu như (tính== &Phép trừ
                printf("Con trỏ tới hàm con");
        khác
                printf("Con trỏ tới các hàm khác");
}
khác
        printf("Con trỏ hàm chưa khai báo");
```

Con trỏ hàm - 6

- Gọi hàm thông qua con trỏ hàm
  - Sử dụng toán tử "*" (trang trọng) nhưng trường hợp này có thể bỏ qua

```
int Sum(int x, int y);
int Phép trừ(int x, int y);

int (*tính toán)(int, int);

tính = Tổng;
int kq1 =(*tính toán)(1, 2);                    // Chính thức
int kq2 = tính toán (1, 2); // Kiểu ngắn
```

Con trỏ hàm - 7

- Truyền tham số làm con trỏ hàm

```
int Sum(int x, int y);
int Phép trừ(int x, int y);
int Tính toán(int x, int y,int (*toán tử)(int,
int))
{
        int kq =(*toán tử)(x, y); // Gọi hàm
        trả lại kq;
}

khoảng trống chính()
{
        int (*toán tử)(int, int) = ∑
        int kq1 = Tính(1, 2, toán tử);
        int kq2 = Tính(1, 2, &Trừ);
}
```

Con trỏ hàm - 8

- Trả về con trỏ hàm

```
int (*GetOperator(mã char))(int, int)
{
        nếu (mã == '+')
                trả lại &Tổng;
        trả lại &Trừ;
}

khoảng trống chính()
{
        int (*toán tử)(int, int) = NULL;
        toán tử = GetOperator('+');
        int kq2 = toán tử(1, 2, &Phép trừ);
}
```

Con trỏ hàm - 9

- Trả về con trỏ hàm

```
typedef (*Nhà điều hành)(int, int);
Nhà điều hànhGetOperator(mã char) {

        nếu (mã == '+')
                trả lại &Tổng;
        trả lại &Trừ;
}

khoảng trống chính()
{
        Toán tử toán tử = NULL;
        toán tử = GetOperator('+');
        int result2 = toán tử(1, 2, &Phép trừ);
}
```

Con trỏ hàm - 10

- Mảng con trỏ hàm

```
typedef (*Nhà điều hành)(int, int);
khoảng trống chính()
{
        int (*array1[2])(int, int); // rõ ràng Nhà điều hành
        mảng2[2];                   // ngầm định

        mảng1[0] = mảng2[1] = ∑ mảng1[1] = mảng2[0] =
        &Phép trừ;

        printf("%d\n", (*array1[0])(1, 2));
        printf("%d\n", array1[1](1, 2)); printf("%d\n",
        array2[0](1, 2)); printf("%d\n", array2[1](1, 2));

}
```

Con trỏ hàm - 11

- Ghi chú

  - Không bỏ sót (*) khi khai báo con trỏ hàm
    - int(*Nhà điều hành)(intx,inty);
    - int*Nhà điều hành(intx,inty);
  - Có thể bỏ qua tên tham số khi trỏ hàm
    - int(*Nhà điều hành)(intx,inty);
    - int(*Nhà điều hành)(int,int);

Chức năng gọi lại - 1

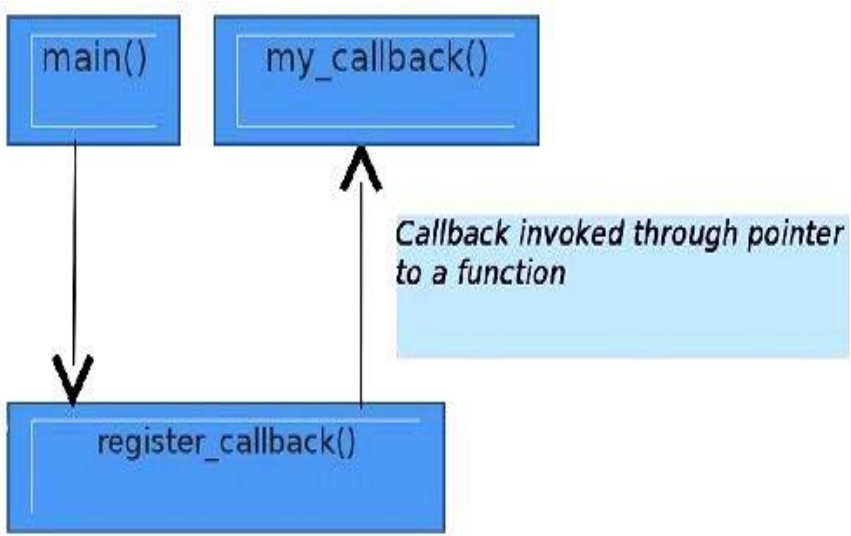
- Ý tưởng

- Chức năng gọi lạilà một hàm được gọi thông qua mộtcon trỏ hàmđó làđược thông qua như một đối sốtừ một phương pháp khác.
- Khi con trỏ đó được dùng để gọi hàm mà nó trỏ tới thì gọi là gọi lại được thực hiện.

Chức năng gọi lại - 2

-ví dụ 1

```
khoảng trống typedef(*gọi lại)(trống rỗng);

trống rỗngmy_callback(trống rỗng) {

   cout<<“bên trong my_callback\n";
}

trống rỗngregister_callback(gọi lại ptr_callback) {

   (*ptr_callback)();
}

intchủ yếu(trống rỗng) {

   gọi lại ptr_my_callback = my_callback;
   register_callback(ptr_my_callback);
   trở lại0;
}
```



Chức năng gọi lại - 3

-Ví dụ 2

```
typedefgấp đôi(*gọi lại)(gấp đôiMột,gấp đôib);

gấp đôiThêm(gấp đôiMột,gấp đôi
b) {trở lạia + b;}

gấp đôiDấu trừ(gấp đôiMột,gấp đôi
b) {trở lạia – b;}

trống rỗngregister_callback(gọi lại ptr_callback,gấp đôiMột,gấp đôib) {
        (*ptr_callback)(a, b);
}

intchủ yếu(trống rỗng) {
        gọi lại ptr_callback = Plus;
        cout << “kết quả = “ << register_callback(ptr_callback, 3, 4) << endl;

        ptr_callback = Dấu trừ;
        cout << “kết quả = “ << register_callback(ptr_callback, 3, 4) << endl;
}
```

Đầu ra:

kết quả = 7

kết quả = -1

Chức năng gọi lại - 4

-Thực hiện gọi lại hàm thành viên C++ tĩnh

- Đây làgiống nhaunhư thực hiện các cuộc gọi lại đếnHàm C++.

```
khoảng trống typedef(*gọi lại)(trống rỗng);

lớp họcLớp học của tôi{
công cộng:
   khoảng trống tĩnhTĩnhCallBack(trống rỗng) {cout<<“bên trong my_callback\n";}
};

trống rỗngregister_callback(gọi lại ptr_callback) {
   (*ptr_callback)();
}

intchủ yếu(trống rỗng) {
   gọi lại ptr_my_callback = &MyClass::StaticCallBack;
   register_callback(ptr_my_callback);
   trở lại0;
}
```

Chức năng gọi lại - 5

-Thực hiện gọi lại hàm thành viên C++ không tĩnh

- Con trỏĐẾNthành viên không tĩnhcầncon trỏ nàycủa một đối tượng lớp được thông qua và viết mộthàm thành viên tĩnhBẰNGmột cái bọc.

```
khoảng trống typedef(*gọi lại)(trống*);

lớp họcLớp học của tôi{
công cộng:
   trống rỗngGọi lại(trống rỗng) {cout<<"bên trong my_callback\n";}
   khoảng trống tĩnhWrapper_To_Call(trống*ptObject) {

         // truyền rõ ràng tới một con trỏ tới MyClass
         MyClass* objA = (MyClass*) ptObject;

        // gọi thành viên
         objA->CallBack();
   }
};
```

Chức năng gọi lại - 6

-Triển khai Lệnh gọi lại cho Hàm thành viên C++ không tĩnh (tiếp theo)

```
trống rỗngregister_callback(trống*ptobject, gọi lại ptr_callback) {

    (*ptr_callback)(ptobject);
}

intchủ yếu(trống rỗng)
{
    MyClass objA;
    gọi lại ptr_my_callback = &MyClass ::Wrapper_To_Call;
    register_callback((void*)&objA, ptr_my_callback); trở lại0;

}
```

Cảm ơn

Hỏi đáp