

# Bài 01

## ÔN TẬP KỸ THUẬT LẬP TRÌNH

### MỤC TIÊU

Hoàn tất bài thực hành này, sinh viên có thể:

- Hiểu và sử dụng kiểu con trỏ trong C++.
- Phân biệt truyền tham biến và truyền tham trị.
- Thao tác đọc/ghi trên tập tin văn bản.
- Hiểu rõ về lập trình đệ quy, viết được các chương trình đệ quy.

### THỜI GIAN THỰC HÀNH

Từ 6-15 tiết, gồm

- Con trỏ: 2-5 tiết
- Truyền tham biến và truyền tham trị: 1-3 tiết
- Thao tác đọc/ghi trên tập tin văn bản: 2-4 tiết
- Lập trình đệ quy: 1-3 tiết

## 1. CON TRỎ

Con trỏ là khái niệm đặc biệt trong C/C++, là loại biến dùng để chứa địa chỉ. Các thao tác với con trỏ lần lượt qua các bước:

- Khai báo biến con trỏ
- Khởi tạo con trỏ dùng cấp phát vùng nhớ
- Truy xuất giá trị ô nhớ từ biến con trỏ
- Hủy vùng nhớ đã xin cấp phát

### 1.1. Khai báo biến con trỏ trong C++

<KieuDuLieu> \*<TenBien>;

Ví dụ:

```
int* pa; // con trỏ đến kiểu int
DIEM *pd; // con trỏ đến kiểu DIEM
```

Để xác định địa chỉ của một ô nhớ: toán tử &

Ví dụ:

```
int a = 1;
int* pa = &a; // con trỏ trỏ đến ô nhớ của biến a
```

### 1.2. Khởi tạo biến con trỏ dùng cấp phát vùng nhớ (cấp phát động)

Sử dụng toán tử new

Ví dụ:

```
int* pInt = new int; // xin cấp phát vùng nhớ cho 1 số nguyên
DIEM *pDiem = new DIEM; // xin cấp phát vùng nhớ cho 1 biến kiểu cấu trúc DIEM
```

Toán tử new còn có thể sử dụng để cấp phát vùng nhớ cho **nhiều** phần tử.

```
int* arr = new int[5]; // xin cấp phát vùng nhớ cho 5 số nguyên
```

**Lưu ý:**

Để kiểm tra cấp phát vùng nhớ thành công hay không, ta dùng con trỏ đặc biệt NULL.

NULL là con trỏ đặc biệt, có thể được gán cho các biến con trỏ của các kiểu dữ liệu khác nhau.

Ví dụ:

```
int* pInt = NULL;
DIEM* pDiem = NULL;
```

đều hợp lệ.

Để kiểm tra cấp phát thành công, ta làm như sau:

```
DIEM* pDiem = NULL; // khai báo con trỏ và gán bằng NULL

pDiem = new DIEM;    // xin cấp phát vùng nhớ

if (pDiem == NULL)   // nếu pDiem vẫn bằng NULL thì xin cấp phát không thành công
    printf("Cap phat khong thanh cong");
```

### 1.3. Truy xuất giá trị ô nhớ từ biến con trỏ

#### 1.3.1. Đối với các kiểu dữ liệu cơ bản (như kiểu int, float, ...)

Để xác định giá trị ô nhớ tại địa chỉ trong biến con trỏ: toán tử \*

Ví dụ:

Với khai báo các biến a, pa

```
int a = 1;

int* pa = &a; // con trỏ trỏ đến ô nhớ của biến a
```

câu lệnh

```
printf("%d\n", *pa);
```

sẽ xuất ra “1”

Giải thích:

```
int a = 1;
```

Với khai báo này, một ô nhớ sẽ được cấp phát và nội dung ô nhớ là 1

```
int* pa = &a;
```

Sau khai báo này, biến pa sẽ giữ địa chỉ ô nhớ vừa được cấp phát cho biến a

Khi đó, \*pa sẽ lấy nội dung của ô nhớ được trỏ đến bởi biến pa, mà biến pa giữ địa chỉ ô nhớ được cấp phát cho biến a.

Vậy \*pa = a = 1.

#### 1.3.2. Đối với các kiểu dữ liệu cấu trúc (như kiểu SINHVIEN, DIEM, ...)

Để truy xuất các thành phần của kiểu cấu trúc, dùng ->

Ví dụ:

Với kiểu cấu trúc DIEM được định nghĩa như sau

```
struct DIEM
{
```

```

        int hoànhDo, tungDo;

    } ;

    DIEM *pDiem = new DIEM;

```

Để truy xuất thành phần dùng

```
pDiem->hoanhDo và pDiem->tungDo
```

#### 1.4. Hủy vùng nhớ đã xin cấp phát

Để hủy vùng nhớ đã xin cấp phát, dùng toán tử delete

Với khai báo

```

int* pa = new int;

int* pb = new int[5];

```

Cách hủy tương ứng sẽ là

```

delete pa;

delete pb[];

```

#### Bài tập (code mẫu: ConTroCoBan)

```

#include <stdio.h>

struct DIEM
{
    int hoànhDo, tungDo;
} ;

void main()
{
    // khởi tạo các biến giá trị

    int a = 1;
    DIEM d;
    d.hoanhDo = 1;
    d.tungDo = 2;

    // khai báo biến con trỏ và trỏ đến vùng nhớ của các biến giá trị đã có
    int *pa = &a;
    int *pb = pa;
    DIEM *pd = &d;

    // xác định địa chỉ ô nhớ: toán tử &
    printf("Địa chỉ ô nhớ: %d \n", &a);

    // truy xuất giá trị ô nhớ từ biến con trỏ: toán tử *
    printf("Giá trị a: %d \n", *pa);

    // truy xuất thành phần trong cấu trúc

    printf("Điểm D: (%d,%d)\n", d.hoanhDo, d.tungDo); //đối với biến giá trị: .
    printf("Điểm D: (%d,%d)\n", pd->hoanhDo, pd->tungDo); // đối với biến con
tro: ->
    delete pd;
}

```

1. Biên dịch đoạn chương trình trên.
2. Nếu lệnh

```
int a = 1;
```

được đổi thành

```
int a = 10;
```

Cho biết giá trị của \*pa.

3. Nếu dòng

```
int *pa = &a;
```

được sửa lại thành

```
int *pa;
```

Cho biết kết quả biên dịch chương trình? Chương trình có báo lỗi khi thực thi không? Nếu có, cho biết tại sao lỗi.

4. Nếu trước dòng

```
printf("Gia tri a: %d \n", *pa);
```

ta thêm dòng code

```
*pb = 2;
```

Cho biết kết quả của lệnh xuất

```
printf("Gia tri a: %d \n", *pa);
```

Giải thích tại sao có kết quả xuất như vậy.

### 1.5. Con trỏ với mảng (cấp phát mảng động)

Cách làm trước đây khi không sử dụng cấp phát động với mảng 1 chiều

```
int a[100]; // xin 100 ô nhớ cho mảng tối đa 100 phần tử
int n;
printf("Nhap so luong phan tu: ");
scanf("%d", &n);
for (int i = 0; i < n; i++)
{
    printf("Nhap a[%d]: ", i);
    scanf("%d", &a[i]);
}
```

Cách làm này có nhiều hạn chế như: cấp phát thừa trong trường hợp n nhập vào < 100 và không cho phép n nhập vào lớn hơn một số lượng định trước được cài đặt trong code (100).

Để cấp phát mảng động (số lượng phần tử cấp phát đúng bằng với n nhập vào và không giới hạn giá trị n), ta làm như sau

```
int n;
printf("Nhap so luong phan tu: ");
scanf("%d", &n);

//khai bao bien con tro a va xin cap phat vung nho chua n so integer
int* a = new int[n];

//dung vong lap de nhap cac gia tri a[i]
```

```
for (int i = 0; i < n; i++)
{
    printf("Nhap a[%d]: ", i);
    scanf("%d", &a[i]);
}
```

Sau khai báo

```
int* a = new int[n];
```

một vùng nhớ chứa n số nguyên sẽ được cấp phát, con trỏ a sẽ chỉ đến phần tử đầu tiên của dãy n số.

Các phần tử của mảng được truy xuất qua toán tử [] như với mảng trước đây đã dùng.

### **Bài tập (code mẫu: ConTroVoiMang)**

Nhập mảng một chiều các số nguyên dùng cấp phát mảng động.

```
#include <stdio.h>

void main()
{
    // MANG 1 CHIEU

    int n;
    printf("Nhap so luong phan tu: ");
    scanf("%d", &n);

    //khai bao bien con tro a va xin cap phat vung nho chua n so interger
    int* a = new int[n];

    //dung vong lap de nhap cac gia tri a[i]
    for (int i = 0; i < n; i++)
    {
        printf("Nhap a[%d]: ", i);
        scanf("%d", &a[i]);
    }

    printf("a[0] = %d\n", a[0]);
    printf("*a = %d\n", *a);

    //xuat cac gia tri a[i]
    for (int i = 0; i < n; i++)
    {
        printf("a[%d] = %d\n", i, a[i]);
    }
}
```

1. Biên dịch đoạn chương trình trên.
2. Nhập vào một vài mảng số nguyên, nhận xét về kết quả của 2 lệnh xuất sau các lần chạy.

```
printf("a[0] = %d\n", a[0]);
printf("*a = %d\n", *a);
```

3. Giải thích tại sao có thể rút ra kết luận ở câu 2.
4. Sửa lại đoạn chương trình trên để nhập vào một mảng số nguyên và xuất ra tổng các số trong mảng đó.

5. Viết chương trình cho phép nhập vào một mảng 2 chiều các số nguyên dùng cấp phát động.

Gợi ý:

Mảng 2 chiều  $m \times n$  các số nguyên được khai báo như sau

```
int** b = new int*[m];
```

trong đó mỗi  $b[i]$  (kiểu  $\text{int}^*$ ) là một mảng một chiều  $n$  số nguyên

```
b[i] = new int[n];
```

## 2. TRUYỀN THAM SỐ: TRUYỀN THAM BIẾN vs TRUYỀN THAM TRỊ

Giả sử ta có đoạn chương trình:

```
void BinhPhuong(int a)
{
    a = a * a;
}
void main()
{
    int a = 2;
    BinhPhuong(a);
    printf("%d", a);
}
```

với hàm BinhPhuong nhận vào 1 tham số kiểu int và tính bình phương của số đó ( $a = a * a$ ).

Ta mong muốn kết quả xuất ra là 4. Tuy nhiên, thực tế kết quả xuất ra lại là 2.

Giải thích:

Khi gọi thực hiện hàm Funtciton(a), giá trị của a sẽ được truyền cho hàm, không phải là bản thân biến a. Do đó, dù câu lệnh

```
a = a * a;
```

được thực hiện và giá trị của a trong hàm BinhPhuong có thay đổi nhưng do ta chỉ truyền giá trị chứ không phải bản thân biến nên khi ra khỏi hàm BinhPhuong(), giá trị của biến a khi thực hiện câu lệnh in ra vẫn là 2.

Cách truyền tham số a vào hàm BinhPhuong như trên gọi là cách **truyền tham trị** (chỉ truyền giá trị vào hàm, các thao tác làm thay đổi giá trị của biến bên trong hàm không ảnh hưởng đến giá trị biến khi kết thúc hàm).

Tuy nhiên trong trường hợp này, ta muốn những thay đổi giá trị biến a trong hàm BinhPhuong vẫn có tác dụng khi ra khỏi hàm. Ta sửa lại đoạn chương trình trên như sau

```
void BinhPhuong(int& a)
{
    a = a * a;
}
void main()
{
    int a = 2;
    BinhPhuong(a);
    printf("%d", a);
}
```

thì kết quả xuất ra là 4.

trong đó cách khai báo

```
void BinhPhuong(int& a) // dùng toán tử &
```

cho biết biến a sẽ được truyền theo kiểu **tham biến** (truyền trực tiếp biến vào hàm, do đó những thay đổi giá trị của biến bên trong hàm sẽ ảnh hưởng đến giá trị biến kể cả khi kết thúc hàm).

**Bài tập (code mẫu: ThamBien\_ThamTri)**

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

Tài liệu hướng dẫn thực hành môn **Cấu trúc dữ liệu và giải thuật**  
HCMUS 2010



```

struct DIEM
{
    int x, y;
};

void TruyenThamTri(int a)
{
    a = a * 10;
}

void TruyenThamBien(int &a)
{
    a = a * 10;
}

void ThamTriConTro(DIEM* d)
{
    d->x = d->x * 10;
    d->y = d->y * 10;
}

void ThamBienConTro(DIEM* &d, DIEM* p)
{
    d->x = d->x * 10;
    d->y = d->y * 10;
    d = p;
}

void main()
{
    // tham tri, tham bien voi bien du lieu
    int a = 1, b = 10;

    printf("a = %d\n", a);

    TruyenThamTri(a);
    printf("a sau ham TruyenThamTri = %d\n", a);

    TruyenThamBien(a);
    printf("a sau ham TruyenThamBien = %d\n", a);

    // bien con tro

    DIEM* d2 = new DIEM;
    d2->x = 5; d2->y = 5;
    printf("Diem d2(%d, %d)\n", d2->x, d2->y);

    ThamTriConTro(d2);
    printf("d2 sau khi goi ham ThamTriConTro: (%d,%d)\n", d2->x, d2->y);
    printf("\n");

    DIEM* d1 = new DIEM;
    d1->x = 5; d1->y = 5;
    d2->x = 5; d2->y = 5;
    printf("Diem d2(%d, %d)\n", d2->x, d2->y);
    ThamBienConTro(d2, d1);
    printf("d2 sau khi goi ham ThamBienConTro: (%d,%d)\n", d2->x, d2->y);
}

```

1. Biên dịch đoạn chương trình trên.

2. Cho biết kết quả của lệnh in

Tài liệu hướng dẫn thực hành môn **Cấu trúc dữ liệu và giải thuật**  
HCMUS 2010

```
printf("a sau ham TruyenThamTri = %d\n", a);
```

và

```
printf("a sau ham TruyenThamBien = %d\n", a);
```

### 3. Cho biết kết quả của lệnh in

```
printf("d2 sau khi gọi ham ThamTriConTro: (%d,%d)\n", d2->x, d2->y);
```

Nhận xét giá trị của d2->x và d2->y có bị thay đổi không? Nếu giá trị của d2->x và d2->y có bị thay đổi, giải thích tại sao khi khai báo hàm

```
void ThamTriConTro(DIEM* d)
```

biến d truyền theo kiểu tham trị (không sử dụng &) nhưng d2->x, d2->y lại bị thay đổi

Gợi ý: d là kiểu con trỏ.

### 4. Cho biết kết quả của lệnh in

```
printf("d2 sau khi gọi ham ThamBienConTro: (%d,%d)\n", d2->x, d2->y);
```

Nhận xét giá trị của d2->x và d2->y có bị thay đổi không? Nếu giá trị của d2->x và d2->y không bị thay đổi, giải thích tại sao khi khai báo hàm

```
void ThamBienConTro(DIEM* &d, DIEM* p)
```

biến d truyền theo kiểu tham biến (sử dụng &) nhưng d2->x, d2->y lại không bị thay đổi

Gợi ý

Thay đổi

```
d1->x = 5; d1->y = 5;
```

thành giá trị khác rồi quan sát kết quả của lệnh in để rút ra nhận xét.