Pointer in Constructor, Destructor

CSC10003 – Phương pháp lập trình hướng đối tượng

Nội dung thực hành lần này sẽ tập trung phân tích và hướng dẫn một số vấn đề về con trỏ và vùng nhớ động trong việc khởi tạo và huy lớp đối tượng

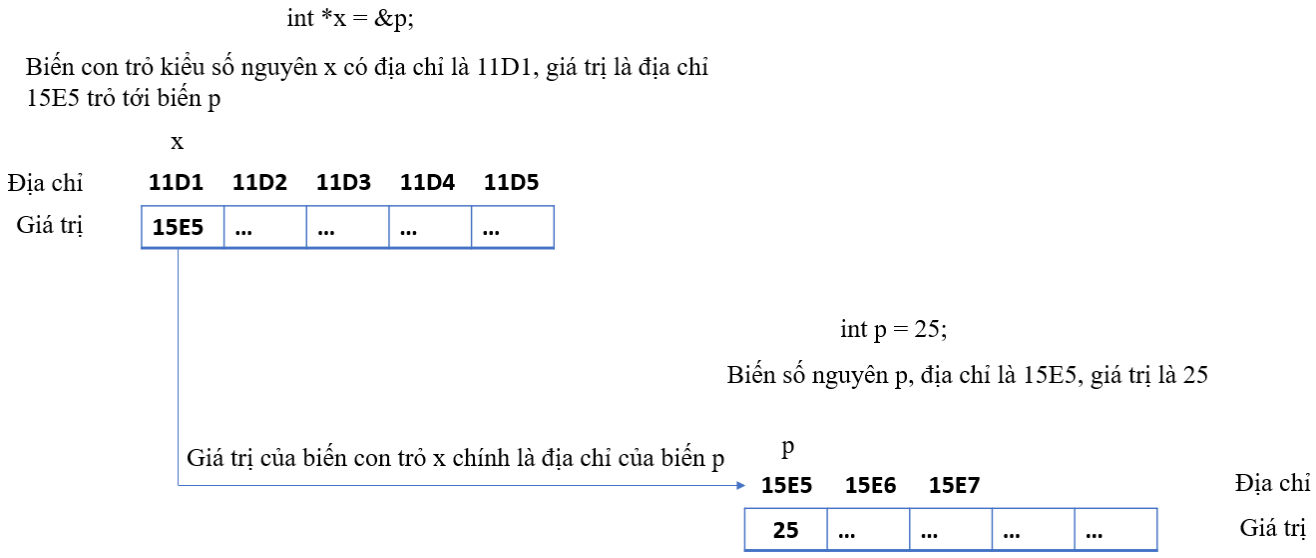
1. Ôn tập về con trỏ

- Khi khai báo biến thông thường, máy tính sẽ tự động cấp phát vùng nhớ dựa trên kiểu dữ liệu khai báo. Vùng nhớ đó chứa giá trị của biến đã được khai báo. Vùng nhớ khi được cấp phát sẽ có địa chỉ riêng và địa chỉ được biểu diễn bằng hệ thập lục phân.

```
int x = 10;
cout << "Gia tri cua x: " << x << endl;
cout << "Dia chi cua bien x: " << &x << endl;
```

```
Gia tri cua x: 10
Dia chi cua bien x: 004FFE90
Press any key to continue . . .
```

- Một con trỏ là một biến dùng để lưu trữ địa chỉ của biến khác. Bản thân biến con trỏ cũng sẽ có địa chỉ riêng của nó, giá trị của biến con trỏ chính là địa chỉ lưu trữ mà biến này trỏ đến, xem mô tả ở hình sau:

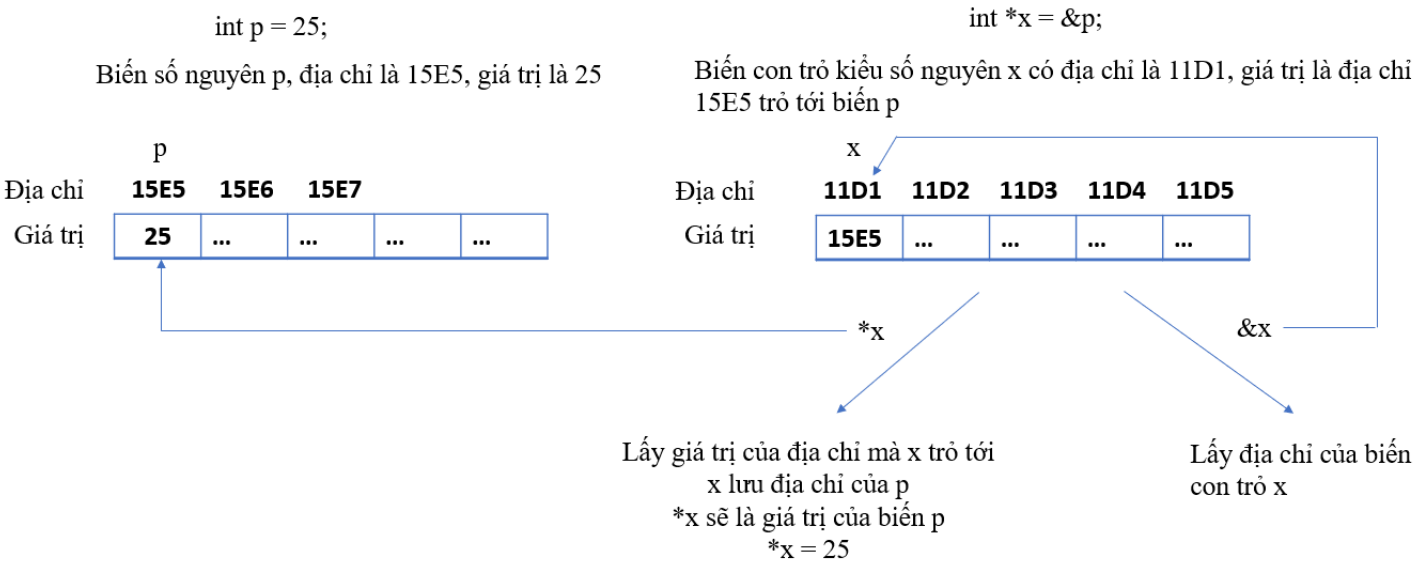


- Để khai báo một biến con trỏ, ta khai báo theo cú pháp:

  kiểu dữ liệu * tên biến;

```
int *x;
```

- Để lấy địa chỉ của một biến (bao gồm cả biến con trỏ), ta dùng toán tử "&"
- Để lấy giá trị của địa chỉ mà biến con trỏ trỏ đến, ta dùng toán tử "*"




```
int p = 25;
int *x = &p;
cout << "Gia tri cua x: " << x << endl;
cout << "Gia tri cua p: " << *x << endl;
cout << "Dia chi cua bien x: " << &x << endl;
```

```
Gia tri cua x: 0059F8A0
Gia tri cua p: 25
Dia chi cua bien x: 0059F894
Press any key to continue . . .
```

- Lưu ý, do biến con trỏ có giá trị là địa chỉ của một biến khác, nên khi gán cần phải lấy địa chỉ của biến bình thường

```
int *x = &p;
```

2. Cấp phát vùng nhớ động

- Trong trường hợp chưa biết được số lượng vùng nhớ cần cấp phát, có thể sử dụng cấp phát động (ví dụ: chưa biết mảng có bao nhiêu phần tử, …). Để cấp phát động một biến con trỏ và một mảng, ta khai báo theo cú pháp:
- Cấp phát động một biến con trỏ:

kiểu dữ liệu * tên biến = new kiểu dữ liệu;

```
int *x = new int;
```

- Cấp phát động một mảng số nguyên

kiểu dữ liệu * tên biến = new kiểu dữ liệu[số lượng phần tử];

```
int *x = new int[10];
```

- Sau khi cấp phát động xong, cần phải xóa vùng nhớ đã cấp phát, nếu không sẽ bị rò rỉ bộ nhớ (Memory Leak). Xóa vùng nhớ đã cấp phát theo cú pháp
  - Xóa vùng nhớ của một biến con trỏ:

delete tên biến;

```
int *x = new int;
delete x;
```

  - Xóa vùng nhớ của một mảng động:

delete[] tên biến;

```
int *x = new int[10];
delete[] x;
```

3. Thuộc tính con trỏ trong constructor và destructor

- Khi cài đặt phương thức tạo lập, cần chú ý đến việc cấp phát vùng nhớ (từ khóa new). Không nên gán trực tiếp biến con trỏ bằng một giá trị vùng nhớ cụ thể!!!
- Cần lưu ý, có bất cứ thuộc tính con trỏ nào thì trong phương thức hủy cần phải gọi lệnh xóa vùng nhớ động (delete), nên cài đặt đầu tiên để tránh quên, dẫn đến phát sinh Memory Leak

```
class PhanSo
{
public:
	int *tuso;
	int *mauso;

public:
	//Phương thức tạo lập mặc định
	PhanSo()
	{
		//Khởi tạo vùng nhớ động
		tuso = new int;
		mauso = new int;
		//Khởi tạo giá trị mặc định ban đầu
		*tuso = 0;
		*mauso = 1;
	}

	//Destructor cần phải hủy vùng nhớ động được tạo
	~PhanSo()
	{
		delete tuso;
		delete mauso;
	}
};
```

- Phương thức tạo lập sao chép khởi tạo đối tượng của một lớp dựa trên việc sao chép một đối tượng sẵn có khác của lớp đó.
- Nếu không cài đặt phương thức tạo lập sao chép, trong ngôn ngữ C++ được trình biên dịch cung cấp mặc định cho lớp đối tượng, được gọi là phương thức tạo lập sao chép mặc định.
  - Việc sao chép mặc định này chính là việc sao chép từng thành phần thuộc tính. (bitwise copy)
  - Sao chép từng thành phần gặp vấn đề khi có thuộc tính con trỏ: chỉ sao chép địa chỉ con trỏ chứ không sao chép nội dung!!! (shallow copy)
- Vậy trong trường hợp thuộc tính có biến con trỏ, bắt buộc phải cài lại phương thức tạo lập sao chép để 3 vấn đề sau:
  - Chỉ sao chép địa chỉ chứ không sao chép nội dung
  - Cả 2 con trỏ đều chỉ đến chung vùng nhớ!!!
  - Phương thức hủy gọi lệnh delete trong khi phương thức tạo lập sao chép mặc định không tạo vùng nhớ mới mà chỉ sao chép địa chỉ vùng nhớ

- Ví dụ cài đặt phương thức tạo lập sao chép với thuộc tính con trỏ

```
//Copy Constructor
PhanSo(const PhanSo& ps)
{
	//Khởi tạo vùng nhớ mới
	tu = new int;
	mau = new int;
	//Sao chép giá trị sang vùng nhớ mới
	*tu = *ps.tu;
	*mau = *ps.mau;
}
```

4. *Bài tập thực hành*

Bài tập 1: Viết chương trình C++ cài đặt lại lớp Phân Số, trong đó thuộc tính của lớp Phân Số được định nghĩa như sau:

```
class PhanSo
{
	int *tuso;
	int *mauso;
};
```

- Xây dựng phương thức tạo lập mặc định, 1 tham số, 2 tham số và sao chép.
- Xây dựng getter, setter cho toàn bộ thuộc tính
- Xây dựng các phương thức cộng, trừ, nhân, chia 2 phân số, lưu ý không làm thay đổi giá trị thuộc tính của cả 2 phân số đầu vào.
- Xây dựng phương thức cho phép nhập và xuất thông tin của lớp Phân Số
  - Xuất theo định dạng tử số / mẫu số
  - Xuất theo định dạng kết quả phép chia của tử số và mẫu số (số thập phân)

Bài tập 2: Bạn đang tập xây dựng trò chơi chiến thuật thời gian thực. Trong đó, bạn phụ trách phần xây dựng kiến trúc và hệ thống cho lớp các tướng chỉ huy của các binh đoàn tộc hệ có trong trò chơi. Bạn trao đổi với bên xây dựng ý tưởng trò chơi và đưa đến việc thống nhất thông tin của một vị tướng bao gồm:

- Tên của vị tướng
- Chỉ số máu
- Chỉ số năng lượng
- Cấp độ của vị tướng
- Một danh sách các chiêu thức cần có của vị tướng đó

Trong đó, một chiêu thức của vị tướng sẽ có các thông tin:

- Tên chiêu thức
- Cấp cần thiết để mở chiêu thức

```
class Skill
{
	string skillName;
	unsigned int skillLevel;
};

class Hero
{
	string heroName;
	unsigned int heroHealth;
	unsigned int heroMana;
	unsigned int heroLevel;
	vector<Skill*> skillList;
};
```

- Sinh viên hãy viết chương trình C++ để cài đặt 2 lớp Hero và Skill theo mô tả trên với đầy đủ các phương thức tạo lập (kể cả sao chép), phương thức hủy, getter, setter, phương thức nhập và xuất thông tin.
- Sinh viên viết thêm các phương thức sau: (tầm vực public)
  - Kiểm tra và xuất ra tất cả các Skill mà Hero có khả năng học được (dựa trên thông tin cấp độ)
  - Cắt bớt n Skill ở cuối danh sách, với n do người dùng nhập vào từ Console. Nếu số lượng nhập lớn hơn so với số lượng Skill có trong danh sách thì xóa toàn bộ danh sách Skill.