**I . Pointer  
1. Explain the result of following program** - ở dòng lệnh Cout đầu tiên cho ra kết quả 3 3 :  
sau khi cấp phát bộ nhớ cho con trỏ ptr\_a thuộc kiểu int 4 byte   
dòng lệnh : \*ptr\_a =3 // gán giá trị con trỏ ptr\_a bằng 3   
và dòng lệnh : ptr\_b = ptr\_a; // con trỏ b và con trỏ a cùng trỏ tới 1 địa chỉ lúc này con trỏ ptr\_b sẽ lấy nội dung của con trỏ ptr\_a tức là \*ptr\_b = \*ptr\_a = 3 ( địa chỉ của ptr\_a giống ptr\_b)  
  
  
-ở dòng lệnh Cout thứ 2 cho ra kết quả 3 9:  
do sau khi cấp phát bộ nhớ mới cho con trỏ ptr\_b thuộc kiểu int 4 byte  
(địa chỉ của ptr\_b đã thay đổi # vs địa chỉ con trỏ ptr\_a)   
dòng lệnh : \*ptr\_b =9 // gán giá trị con trỏ ptr\_b bằng 9 nhưng điều này sẽ ko làm thay đổi giá trị của \*ptr\_a bởi vì lúc này con trỏ ptr\_b thực hiện thay đổi trên bộ nhớ mới được cấp phát riêng.  
  
-ở dòng lệnh Cout thứ 3 cho ra kết quả 3 3:  
dòng lệnh \*ptr\_b = \* ptr\_a // gán nội dung của con trỏ a cho con trỏ b nên   
\*ptr\_b = \*ptr\_a =3 ( địa chỉ của ptr\_a khác ptr\_b)  
  
-ở dòng lệnh Cout thứ 4 cho ra kết quả 3 3:  
vì sau khi delete ptr\_a // sau khi chúng ta giải phóng vùng nhớ cho con trỏ ptr\_a (vùng nhớ của ptr\_a và ptr\_b khác nhau ) con trỏ a lúc này vẫn còn trỏ tới địa chỉ cũ nhưng ko có giá trị tại vùng nhớ bị thu hồi  
dòng lệnh ptr\_a = ptr\_b // lúc này con trỏ a trỏ tới địa chỉ khác là địa chỉ của con trỏ b nên lúc này con trỏ a sẽ mang giá trị \*ptr\_a = \*ptr\_b =3 (địa chỉ ptr\_a giống địa chỉ ptr\_b)  
  
-ở dòng lệnh Cout thứ 5 cho ra kết quả 00055508 3 :  
ở dòng lệnh ptr\_c = &ptr\_a // ở đây con trỏ ptr\_c trỏ đến địa chỉ của con trỏ ptr\_a  
nên nội dung \*ptr\_c sẽ trả về giá trị của con trỏ ptr\_a 00055508  
còn \*\*ptr\_c là con trỏ trỏ tới con trỏ \*ptr\_c nên nội dung của \*\*ptr\_c chính là \*ptr\_a =3

**2. Detect and solve problems of following program**  
#include <iostream>

using namespace std;

void main()

{

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

int \*p = a;

int \*p2 = new int;

//delete p;

delete a;

delete p2;

}  
  
  
lệnh delete p ; giải phóng vùng nhớ cho con trỏ p trong trường hợp này sẽ bị lỗi . bởi vì trong hàm con trỏ p chưa được cấp phát vùng nhớ.   
  
**3. Using pointer:**

Why should we use delete?   
  
- bởi vì delete là một toán tử được sử dụng để phá hủy các đối tượng mảng và con trỏ được tạo bởi biểu thức new.giúp chúng ta giải phóng được vùng nhớ mà những biến con trỏ hay mảng đang quản lí khi ko cần thiết dùng đến nó nữa. Giúp chúng ta giải phóng bộ nhớ khỏi heap. Khi dùng delete thì con trỏ không bị hủy con trỏ trên stack và vùng nhớ trên heap vẫn giữ nguyên . Chỉ mối liên kết giữa con trỏ đến ô nhớ bị hủy. giá trị của con trỏ mà ta nhận được sau khi delete thực tế chỉ là nó đang tạm trỏ đến 1 vùng nhớ bất kì nào đó không xác định. Nên sau khi xóa chúng ta nên cho con trỏ trỏ đến null.  
  
When we use delete?  
  
- Dùng delete khi những biến con trỏ hay mảng đang quản lí ko cần dùng đến nữa sau khi mình dùng toán tử new. Để trả lại vùng nhớ đã cấp phát trên heap cho chương trình có thể sử dụng.  
  
Difference between delete and delete[]. Write a demo?  
  
- Khi chúng ta muốn giải phóng bộ nhớ được cấp phát cho một con trỏ trỏ tới một đối tượng với toán tử new . ( chỉ có chính xác một thể hiện ) thì từ khóa delete sẽ được dùng. Chỉ có một đối tượng duy nhất được tạo ra với new.  
Trong khi delete[] sẽ gọi hàm hủy trên mọi phần tử mà mảng phân bổ trên heap với toán tử new [] ( ví dụ như mảng động ) .Một mảng các đối tượng được tạo ra với new[]  
  
Ví dụ demo :   
- Dùng delete để giải phóng bộ nhớ  
 ( int \* p;

p=new int;

//giải phóng bộ nhớ cho p

delete p; )  
  
nhưng trong trường hợp này nếu ta dùng con trỏ trỏ tới 1 mảng gồm 5 số nguyên thì để giải phóng bộ nhớ ta dùng delete[]  
( int \* p= new int[5];  
 delete []p; )  
Nếu ở delete []p ta thay bằng delete p . thì ta vẫn có thể giải phóng bộ nhớ . nhưng nó chỉ giải phóng bộ nhớ phần tử đầu tiên.  
  
  
**4. Given below code**1. \*p1 == 10  
2. \*p2 == 0x100  
3. \*(\*p2) == 10