**W04M01 - Con trỏ**

**1. Khái niệm cơn bản:**  
- Các kiểu dữ liệu của C/C++ đều xác định trước**số Bytes** dùng để chứa kiểu đó.  
- Mỗi khi khai báo một biến, hằng có kiểu, trình biên dịch sẽ xác định **1 khu vực trong bộ nhớ**có **độ lớn** bằng số Bytes của kiểu cần khai báo nhằm chứa dữ liệu. Ví dụ: khi khai báo: int n; lúc này một vùng nhớ chiếm 4 Bytes sẽ được cấp phát trong bộ nhớ của chương trình (vùng heap), và gán cho vùng nhớ này tên là n, với câu lệnh, n = 3; vùng nhớ vừa được khai báo được đặt lại giá trị là 3, 3 được gọi là giá trị của vùng nhớ n.  
- Vậy làm sao để biết vùng nhớ mang tên là n đang ở **VỊ TRÍ**nào trong bộ nhớ? Ta sử dụng một kiểu dữ liệu đặc biệt gọi là con trỏ (pointer), giá trị của con trỏ chính là **ĐỊA CHỈ** của vùng nhớ mà nó trỏ đến.  
- Độ lớn của con trỏ tùy thuộc vào hệ điều hành, với Hệ điều hành đánh địa chỉ 32 bits con trỏ có độ lớn là 4 Bytes = 32 bits, với Hệ điều hành 64 bits thì con trỏ có độ lớn là 8 Bytes = 64 bits.  
  
**2. Khai báo và sử dụng con trỏ:**  
- Cú pháp khai báo:  
<kiểu dữ liệu> **\***<tên biến>;  
Ví dụ:  
int \*a;// khai báo con trỏ để trỏ đến các biến kiểu int và đặt tên là a;  
double \*d;// Khai báo con trỏ để trỏ đến các biến kiểu int và đặt tên là d;  
- Gán địa chỉ của biến cho con trỏ hay trỏ con trỏ đến vị trí của 1 biến nào đó  
<contrỏ> = **&** <tên biến>;  
- Lấy giá trị/nội dung của biến mà con trỏ đang trỏ đến  
**\*** <tên con trỏ>;  
Ví dụ:  
1. double n = 3.0;  
2. double \*p;  
3. p = &n;  
4. cout<<p<<" "<<\*p;  
  
Kết quả: 0x22ff08 3  
  
**Giải thích:**  
Câu lệnh 1. Khai báo một biến kiểu int đặt tên là n và có giá trị là 3; biến này có độ lớn là 8 Bytes.  
Câu lệnh 2. Khai báo một biến con trỏ kiểu double, đặt tên là p, với hệ điều hành hiện tại 32 bits, con trỏ có độ lớn là 4 Bytes, thông thường p sẽ mang giá trị mặc định 0x0.  
Câu lệnh 3. Trỏ con trỏ đến n, phép toán **&** cho biết lấy địa chỉ của biến nằm sau nó, do đó &n sẽ lấy địa chỉ của biến n và gán vào p.  
Câu lệnh 4. xuất ra tuần tự, giá trị đang lưu trong con trỏ và giá trị đang lưu tại địa chỉ mà con trỏ trỏ đến. Giá trị đầu có thể khác nhau, nhưng giá trị thứ 2 sẽ là 3.0.  
**0x22ff08** là cách thể hiện một con số ở hệ thập lục phân (hexadecimal), mỗi số trong dãy số sau 0x chiếm 4 bits, dãy số này có 6 số tương đương 24 bits, vậy tại sao nói con trỏ kiểu double ở windows 32 là 32 bits? (cái đó sẽ học vào tuần tới).  
  
  
**3. Một số ghi chú khác**  
- Khi khai báo một mảng, các thành phần của mảng này sẽ được cấp phát khác nhau, do đó nếu con trỏ đang ở vị trí nào đó trong mảng, tăng con trỏ lên 1 đơn vị, tương đương ta đang di chuyển đển phần tử kế tiếp của mảng, giảm đi 1 đơn vị, tương đương di chuyển đến phần tử phía trước của mảng.  
Ví dụ:  
1. int a[5]={1,6,2,3,7};  
2. int \*p;  
3. p = a;  
4. cout<<\*p<<" ";  
5. p++;  
6. cout<<\*p<< " ";  
7. p= a+4;  
8. cout<<\*p<<" ";  
9. cout<<\*(a+4);  
10. cout <<\*a+4;  
  
**Kết quả:**  
1 6 7 7 5  
  
**Giải thích:**  
1. khai báo 1 mảng kiểu int gồm 5 phần tử tuần tự là 0:1, 1:6, 2:2, 3:3 và 4:7.  
2. khai báo con trỏ kiểu int tên là p;  
3. Trỏ p đến vị trí của a **đang trỏ đến**.  
4. Xuất ra giá trị đang chứa tại vị trí trong bộ nhớ mà p trỏ đến, ở đây là phần tử **đầu tiên** của mảng a, là 1  
5. và 6. Tăng con trỏ p lên 1, thực chất lúc này con trỏ p sẽ di chuyển 1 khoảng tương đương với 4 Bytes trong bộ nhớ, lúc này p đang trỏ đến phần tử thứ 2, do đó kết quả là 2.  
7.  và 8. Trỏ p đến vị trí (a+4) là vị trí a đang trỏ đến + 4\*4 Bytes, nghĩa là vị trí thứ 4 của mảng a, kết quả lá 7  
9. Tương tự như vậy, dòng lệnh này cho ta thấy, **thực chất khi khai báo một mảng, tên của mảng đó chính là con trỏ trỏ đến đầu mảng**.  
9. và 10. **Cần cẩn thận** với phép lấy giá trị tại địa chỉ, \*(a+4) nghĩa là lấy giá trị tại vị trí a[4], là 7, còn \*a + 4 nghĩa là lấy giá trị tại a[0] + 4 = 1 + 4 = 5  
  
**4. Bộ nhớ động dùng operator new và delete**  
- Tạo mới một vùng nhớ và trỏ con trỏ đến vùng nhớ đó:  
<con trỏ> =**new** <ten kiểu>;  
  
- Tạo mới một vùng nhớ, xác định giá trị ban đầu và trỏ con trỏ đến vùng nhớ đó:  
<con trỏ> = **new** <ten kiểu>**(giá trị)**;  
  
- Tạo mới một mảng vùng nhớ mới, trỏ con trỏ đến vị trí đầu của vùng nhớ này:  
<con trỏ> = **new** <ten kiểu>[**giá trị]**;  
  
**Nếu tạo được vùng nhớ thì con trỏ sẽ mang giá trị là địa chỉ đầu tiên của vùng nhớ, nếu không, con trỏ sẽ mang giá trị 0x0/NULL.**  
- Xóa vùng nhớ được trỏ tới bởi con trỏ:  
**delete**<tên con trỏ>;  
- Xóa một mảng được tạo ra bởi con trỏ  
**delete** **[]**<tên con trỏ>;  
  
**5. Bộ nhớ động dùng malloc, realloc**  
- **malloc**dùng để tạo mới một vùng nhớ động trong lúc chương trình đang dạy có kích thước **N Bytes**  
<con trỏ> = **(casting type \*)**malloc(**N\*sizeof(type)**);  
  
- **calloc** dùng để tạo mới một vùng nhớ động trong lúc chương trình đang dạy có kích thước **N Bytes, sau đó khởi tạo các giá trị trong vùng nhớ đó bằng 0;**  
<con trỏ> = **(casting type \*)**calloc(**N\*sizeof(type)**);  
  
  
- **realloc**dùng để thu nhỏ hoặc kéo dài một vùng nhớ động trong lúc chương trình đang dạy có kích thước **N Bytes**  
<con trỏ> = **(casting type \*)**realloc(**N\*sizeof(type)**);  
  
**malloc, calloc, realloc** sẽ tạo ra một vùng nhớ với kích thước theo yêu cầu, con trỏ sẽ trỏ đến vị trí đầu của vùng nhớ đó, nếu không tạo được thì con trỏ sẽ mang giá trị NULL, **khi dùng malloc cần kiểm tra xem có tạo được vùng nhớ không.**  
  
- Xóa vùng nhớ được tạo ra bởi **malloc, calloc, realloc.**  
free(void \*ptr);  
  
**Ghi chú:**Khi khai báo vùng nhớ động thì nhớ free sau khi sử dụng, C/C++ sẽ không tự xóa các vùng nhớ này, nếu tạo quá nhiều có thể gây ra lỗi memory overload làm chương trình bị break hoặc chạy không chính xác.  
  
  
**6. So sánh giữa new và malloc/calloc/realloc**  
- new là một operator trong khi malloc là một hàm trong <stdlib.h>  
- new tạo ra vùng nhớ xong sẽ gọi hàm contructor/hàm khởi tạo của kiểu đó, malloc chỉ tạo vùng nhớ không gọi hàm contructor.  
- new là operator nên nhanh hơn mallloc là một hàm.  
- new sẽ trả về con trỏ đúng như kiểu khi gọi, malloc trả về con trỏ không kiểu (void \*) nên phải có (casting type\*).