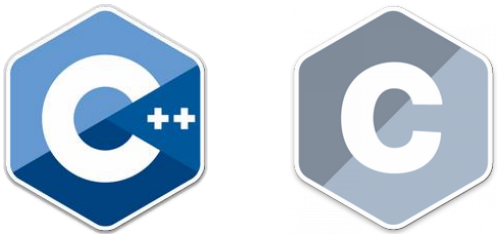
CON TRỎ

Phần 2 – Bộ nhớ RAM



[2017 – 04 – 01]

Biên soạn bởi: Nguyễn Trung Thành
https://www.facebook.com/abcxyztcit

“ Học từ cái đáy của thế giới đi lên là cách duy nhất bạn trở thành master. ”

Nguyễn Trung Thành

Mục lục

A. Giới thiệu

1. Điều kiện để học được tài liệu này

- Có kiến thức nền tảng vững chắc với ngôn ngữ C (hoặc C++).
- Hiểu biết cơ bản về máy tính, bộ nhớ RAM.

2. Dẫn nhập

Khỏi cần dẫn nhập.

B. Bộ nhớ RAM

Phần trước chỉ là giới thiệu con trỏ mà thôi, phần này mới bắt đầu đi sâu vào 1 chút xíu này. Trước khi bắt đầu thì chúng ta làm quen với khái niệm "byte" nhé.

1. Kiểu dữ liệu byte

Khi học cơ bản về máy tính, chúng ta được biết rằng 1 byte = 8 bits.

Khi học sâu hơn về lập trình, ta hiểu đơn giản byte là khái niệm đo lường độ lớn bộ nhớ.

Ví dụ:

- Kiểu int (hoặc unsigned int) có độ lớn là 4 bytes.
- Kiểu short (hoặc unsigned short) có độ lớn là 2 bytes.
- Kiểu char (hoặc unsigned char) có độ lớn là 1 byte.

Bạn hiểu vậy là được.

Bạn có bao giờ nghe người ta nói: máy tính này có RAM 4GB, máy tính kia có RAM 8GB…

- RAM 4GB (4 giga bytes) tức là RAM có độ lớn khoảng 4 tỷ bytes.
- RAM có độ lớn 4GB sẽ chứa được 4 tỷ bytes dữ liệu.
- RAM càng lớn, càng lưu trữ được nhiều dữ liệu.

2. Địa chỉ bộ nhớ và RAM

RAM là bộ nhớ chính của máy tính.

Bạn thử nghĩ xem, khai báo một số nguyên `int n` chiếm 4 bytes, so với 4 tỷ bytes của RAM thì quả thật quá là bé bỏng.

- RAM 4GB = 4 tỷ bytes.
- Số nguyên int có độ lớn 4 bytes.

➔ Xét về mặt lý thuyết, RAM 4GB có thể chứa được 1 tỷ số nguyên kiểu int.

```
int main()
{
	int a0;
	int a1;
	int a2;
	int a3;
	...
	int a999999999;

	return 0;
}
```

} 1 tỷ số nguyên

Để quản lý việc lưu trữ RAM 4GB, người ta đánh số thứ tự các byte,
từ byte 0 đến byte 3 tỷ 999999999.

RAM, cũng như mảng 1 chiều vậy, bản chất RAM 4GB chính là mảng 1 chiều 4 tỷ phần tử, mỗi phần tử có kiểu là byte.

```
byte RAM[4000000000];
```

hoặc

```
unsigned char RAM[4000000000];
```

Nói lý thuyết hơn, RAM gồm các ô nhớ, mỗi ô nhớ có kích thước 1 byte.

Với mảng một chiều a có 4 tỷ phần tử, thì việc ta ghi a[200] sẽ hiểu là ta truy xuất giá trị tại "chỉ số 200 của mảng a".

Còn với RAM, ta cần phải đọc là "truy xuất giá trị tại ĐỊA CHỈ thứ 200 của bộ nhớ RAM".

OK, xong phần 2 này bạn đã hiểu hơn về RAM nhé. Trong C/C++ không có kiểu byte đâu, chỉ có kiểu char hoặc unsigned char tương ứng với 1 byte mà thôi.

Khi đã ok phần 2 này thì chúng ta bắt đầu bước vào phần 3 hiểu rõ tường tận vấn đề nhaaaa.

 Sẽ rất là thú vị đây.

3. Câu chuyện người nông dân và Nhà nước

Có một vùng đất màu mỡ tuyệt đẹp được quản lý bởi Nhà nước.

Có nông dân đến vùng đất này sinh sống. Họ thuê đất của Nhà nước để làm ăn. Trên vùng đất này, họ có thể trồng trọt, tưới cây, chăn nuôi. Sau một thời gian, Nhà nước thu hồi lại đất.

Câu chuyện rất ngắn gọn, lãng xẹt nhảm nhí nhưng chứa đầy hàm ý thâm sâu bí hiểm.

- Nhà nước: hệ điều hành (Windows 10, Windows 8.1, Ubuntu,…).
- Đất: bộ nhớ RAM của máy tính.
- Người nông dân: chương trình.

Từ đó ta suy ra…

- ✓ Người nông dân thuê đất: chương trình xin xỏ bộ nhớ máy tính (RAM).
- ✓ Người nông dân thuê 4 ô đất để làm ăn: chương trình xin xỏ hệ điều hành cung cấp 4 bytes trên RAM để lưu trữ giá trị nào đó (ví dự như lưu trữ 1 số nguyên kiểu int có độ lớn 4 bytes).
- ✓ Người nông dân trồng trọt, tưới cây, chăn nuôi trên mảnh đất của mình: chương trình gán giá trị cho 1 biến, lấy giá trị của 1 biến, sử dụng biến đó…

Hoặc cụ thể hơn…

Khi chương trình khai báo 1 biến số nguyên

```
int n;
```

Chương trình xin xỏ hệ điều hành cung cấp **4 bytes liên tục trên RAM** để lưu trữ số nguyên n. Đây là hành động "thuê đất từ Nhà nước" (mượn RAM từ Hệ điều hành cung cấp).

Khi chương trình chạy xong tức là người nông dân đã hết thời hạn thuê mượn đất, họ phải trả lại đất cho Nhà nước. Vì vậy nên Nhà nước sẽ thu hồi lại đất.

→ Hệ điều hành thu hồi bộ nhớ, lấy lại 4 bytes đã cung cấp cho chương trình (để lưu biến n).

Bạn có bao giờ nghĩ đến trường hợp: sau khi nhà nước thu hồi đất của nông dân A, nhà nước lại tiếp tục cho nông dân B thuê lại chính vùng đất đó ???

Trong quá khứ, người nông dân A cày cấy, sử dụng đất, đất sẽ mang 1 giá trị rác nào đó (ai mà biết được đâu).

Khi người nông dân B nhận đất thì họ sẽ thấy vùng đất này chứa đầy rác, họ phải dọn dẹp lại vùng đất này thì mới yên tâm sử dụng.

Người nông dân B

```
int main()
{
	int n;
	printf("%d", n); // hoặc cout << n; với C++

	return 0;
}
```

Bạn nhìn vào đoạn code trên, bạn có đoán được chương trình sẽ in ra cái gì không ? Chắc chắn là không thể. Khi lập trình cơ bản chúng ta đều biết được điều này. Bây giờ bạn đã hiểu vì sao rồi chứ ? Biến n ở đoạn code trên mang giá trị rác, giá trị rác này do chương trình khác sử dụng trong quá khứ nên chúng ta không xác định được.

→ Thói quen lập trình tốt là phải khởi tạo giá trị cho biến (dọn dẹp vùng đất vừa xin xỏ được).

Nào, bây giờ ta sang phần sau, đi chuyên sâu chi tiết hơn nữa nhé.

4. Bản chất của việc khai báo biến

Nhắc lại: bộ nhớ RAM là dãy các ô nhớ (mỗi ô nhớ là 1 byte), RAM là mảng 1 chiều các byte.

Xét câu lệnh khai báo 1 số nguyên (ghi chú: kiểu int có độ lớn là 4 bytes).

```
int n;
```

Chương trình sẽ xin xỏ hệ điều hành dành ra một vùng trống 4 bytes không ai sử dụng để lưu trữ biến n.

Ban đầu, vùng nhớ RAM có thể có ngữ cảnh như hình dưới.

Giả sử hệ điều hành nhận thấy từ byte 402 tới byte 405 là đất trống không có ai sử dụng, mà lại ***vừa đủ 4 bytes*** nữa. Hệ điều hành quyết định cấp quyền sử dụng vùng đất này cho chương trình của mình.

byte 0	byte 1	byte 2	…	byte 400	byte 401	byte 402	byte 403	byte 404	byte 405	byte 406	…	…

dữ liệu của chương trình khác | vùng đất trống màu mỡ | dữ liệu của chương trình khác

Và kết quả là hình dưới.

byte 0	byte 1	byte 2	…	byte 400	byte 401	byte 402	byte 403	byte 404	byte 405	byte 406	…	…
						biến n						

dữ liệu của chương trình khác | dữ liệu chương trình của mình | dữ liệu của chương trình khác

Lúc này ta nói rằng: **biến n có địa chỉ là 402** trong bộ nhớ RAM.

Wow, vậy nếu ta `printf("%d", &n)` thì sẽ in ra số 402 trên màn hình.

Giả sử ta khởi tạo giá trị cho biến n luôn thì sao nhỉ ?

```
int n = 240;
```

byte 0	byte 1	byte 2	…	byte 400	byte 401	byte 402	byte 403	byte 404	byte 405	byte 406	…	…
						240	0	0	0			

dữ liệu của chương trình khác | dữ liệu chương trình của mình | dữ liệu của chương trình khác

Nếu `int n = 255` thì sao ?

byte 0	byte 1	byte 2	…	byte 400	byte 401	byte 402	byte 403	byte 404	byte 405	byte 406	…	…
						255	0	0	0			

dữ liệu của chương trình khác | dữ liệu chương trình của mình | dữ liệu của chương trình khác

Ghi chú: kiểu byte lưu được giá trị từ 0 đến 255.

Nếu `int n = 256` thì sao ?

byte 0	byte 1	byte 2	…	byte 400	byte 401	byte 402	byte 403	byte 404	byte 405	byte 406	…	…
						0	1	0	0			

dữ liệu của chương trình khác | dữ liệu chương trình của mình | dữ liệu của chương trình khác

Nếu `int n = 257` thì sao ?

byte 0	byte 1	byte 2	…	byte 400	byte 401	byte 402	byte 403	byte 404	byte 405	byte 406	…	…
						1	1	0	0			

dữ liệu của chương trình khác | dữ liệu chương trình của mình | dữ liệu của chương trình khác

Nếu `int n = 258` thì sao ?

byte 0	byte 1	byte 2	…	byte 400	byte 401	byte 402	byte 403	byte 404	byte 405	byte 406	…	…
						2	1	0	0			

dữ liệu của chương trình khác | dữ liệu chương trình của mình | dữ liệu của chương trình khác

Nếu `int n = 259` thì sao ?

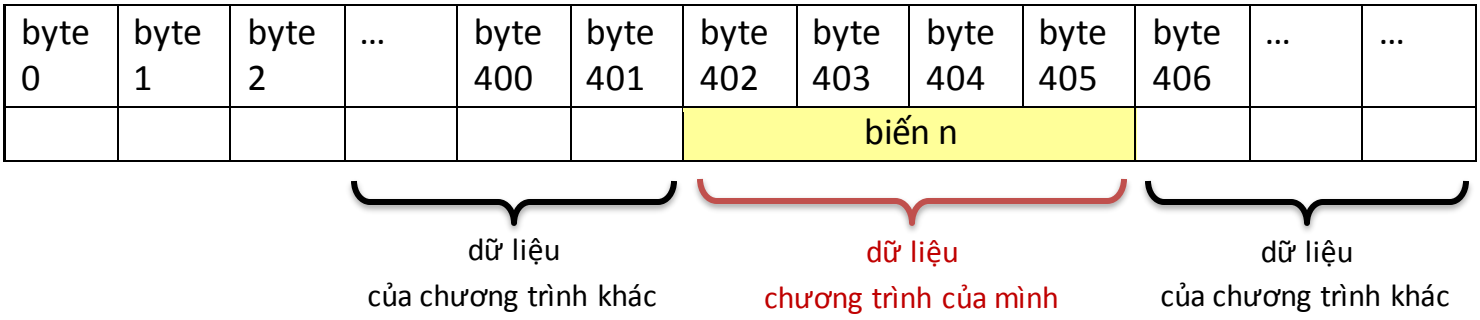
byte 0	byte 1	byte 2	…	byte 400	byte 401	byte 402	byte 403	byte 404	byte 405	byte 406	…	…
						3	1	0	0			

dữ liệu của chương trình khác | dữ liệu chương trình của mình | dữ liệu của chương trình khác

Đấy, cứ thế mà bạn hiểu được rõ nhé ☺.

Ghi chú: các ví dụ trong tài liệu sử dụng Little Endian (bạn chưa cần hiểu lúc này).

Nào, bây giờ ta cùng nhau chiêm nghiệm rút ra bài học nhé.



Trong ngôn ngữ C/C++, toán tử & có nhiều công dụng, 1 trong những công dụng đó là dùng để lấy địa chỉ của biến.

Giả sử như ta `printf("%d", &n);` thì sẽ in ra số 402 (vì địa chỉ của n là 402).

Giả sử ta `scanf("%d", &n)` thì chuyện gì xảy ra ?

- Người dùng sẽ nhập vào giá trị cho biến n, giả sử nhập vào số 93 đi nha.
- Hàm scanf sẽ đi đến địa chỉ lưu biến n (là 402 đó), sau đó sẽ gán giá trị tương ứng vào bộ nhớ.
    - Cụ thể: gán giá trị 93 vào byte số 402, sau đó gán giá trị 0 vào các byte 403, 404, 405.

Nếu từ đầu đến giờ bạn nắm được hết thì xin chúc mừng bạn ☺. Bạn đã có 1 nền tảng tuyệt vời để bắt đầu học con trỏ rồi.

C. Con trỏ

1. Định nghĩa

Các biến trong chương trình, thông thường sẽ lưu các giá trị số nguyên (biến int), lưu số thực (biến float, double), lưu kí tự (biến char).

Khi học xong phần trên, bạn sẽ dần nhận ra rằng bạn cần quan tâm đến địa chỉ trong RAM nhiều hơn. Như vậy ta cần một loại biến mà có thể lưu được địa chỉ trong RAM.

ĐÓ LÀ CON TRỎ.

Cú pháp khai báo con trỏ:

Tên kiểu dữ liệu + dấu sao + tên biến.

Ví dụ 1:

```
int *pi;
```

Khai báo biến con trỏ tên là pi. Biến pi là một biến có kiểu int*
Vì có kiểu int* nên biến pi có thể lưu được địa chỉ các số nguyên.

```
char *px;
```

Khai báo biến con trỏ tên là px. Biến px là một biến có kiểu char*
Vì có kiểu char* nên biến px có thể lưu được địa chỉ các kí tự.

```
PhanSo *a;
```

Khai báo biến con trỏ tên là a. Biến a là một biến có kiểu PhanSo*
Vì có kiểu PhanSo* nên biến a có thể lưu được địa chỉ các PhanSo (struct PhanSo).

Ví dụ 2:

```
int x = 13;
int *px = NULL; // khai báo con trỏ px, px không trỏ vào cái gì hết
px = &x; // px lưu địa chỉ của biến x, tương đương px trỏ đến x
```

Ví dụ 3:

```
int x = 13;
int *px = &x; // khai báo con trỏ px và khởi tạo px lưu địa chỉ của biến x
```

px có giá trị là địa chỉ của biến x.

Ví dụ biến x nằm ở địa chỉ 402 trong RAM.

Như vậy px có giá trị là 402.

Ví dụ 4:

```
int x = 13;
float *px = &x; // sai, không biên dịch được vì khác kiểu dữ liệu
```

Lệnh scanf nhận vào ĐỊA CHỈ của biến, vì vậy nên ta có 2 cách sử dụng scanf:

```
int x = 13;
int *px = &x; // px lưu địa chỉ của x

scanf("%d", &x); // CÁCH 1

scanf("%d", px); // CÁCH 2
```

2. Con trỏ lưu địa chỉ để làm gì ?

Ở phần trước bạn cũng đã học 1 chút rồi, bây giờ mình sẽ giải thích kỹ hơn nhé.

Giả sử biến n nằm tại địa chỉ là 400.

byte 0	byte 1	byte 2	...	byte 398	byte 399	byte 400	byte 401	byte 402	byte 403	byte 404	...	...
						biến n						

dữ liệu của chương trình khác

dữ liệu chương trình của mình

dữ liệu của chương trình khác

Cho đoạn code sau:

```
int n = 27;
int *p = &n; // p = 400
```

Nhờ vào việc lưu địa chỉ của n nên con trỏ p có thể gián tiếp quản lý biến n.

```
// giống như printf("%d", n)
printf("%d", *p); // in ra 27


// giống như scanf("%d", &n);
scanf("%d", p);
```

Từ đây, bạn có 2 cách in ra giá trị của n và 2 cách nhập giá trị của biến n.

Nhìn lại ví dụ trên, khi sử dụng toán tử * ở phía trước con trỏ có nghĩa là bạn đang ***lấy giá trị mà con trỏ trỏ đến***.

Giả sử ta có con trỏ p có kiểu int*.

Dễ dàng suy ra p sẽ lưu địa chỉ của 1 biến số nguyên int (kiểu int có độ lớn là 4 bytes).

Khi ta `printf("%d", *p)` tức là…

Tìm đến địa chỉ của số nguyên int, lấy 4 bytes ra và cấu thành nên giá trị của x → in ra màn hình.

Giả sử ta có con trỏ p có kiểu double* (ghi chú: kiểu double có độ lớn là 8 bytes).

Dễ dàng suy ra p sẽ lưu địa chỉ của 1 biến số thực double.

Khi ta `printf("%d", *p)` tức là…

Tìm đến địa chỉ của số thực double, lấy 8 bytes ra và cấu thành nên giá trị của y → in ra màn hình.

Nói tóm lại, khi ta dùng toán tử * ở phía trước tên biến con trỏ là ta đang sử dụng phép "lấy giá trị" (dereference).

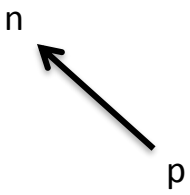
3. Ghi nhớ

Con trỏ là biến dùng để lưu trữ địa chỉ.

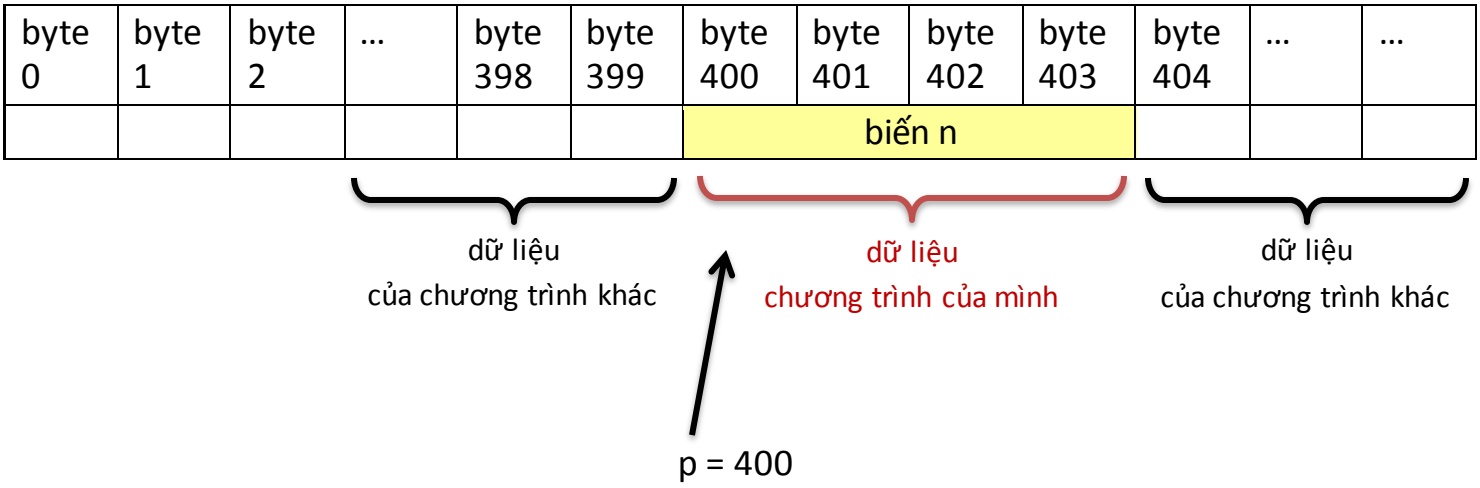
Khi ta cho con trỏ p lưu địa chỉ của biến x, ta nói rằng:

- p trỏ đến x (hoặc: p tham chiếu đến x).
- p có thể gián tiếp quản lý biến x (thông qua phép toán *).

Khi p lưu địa chỉ của n tức là p trỏ đến n, ta dùng hình minh họa như ở dưới:



Minh họa chi tiết hơn:



D. Các phép toán với con trỏ

1. Phép lấy giá trị (dereference)

Chính là phép toán * mà mình đã giải thích kĩ ở phần trước đó bạn.

C	C++

`#include <stdio.h>`
 	
`int main()`
`{`
`  int x = 27;`
`  int *p = &x;`
 	
`  // giống như printf("%d", n)`
`  // in ra 27`
`  printf("%d", *p);`
 	
`  return 0;`
`}`
	`#include <iostream>`
	`using namespace std;`
	
	`int main()`
	`{`
	`  int x = 27;`
	`  int *p = &x;`
	
	`  // giống như cout << n`
	`  // in ra 27`
	`  cout << *p;`
	
	`  return 0;`
	`}`

Ghi chú: có rất nhiều bạn bị rối loạn với dấu sao *.

Bạn cần phân biệt rõ 2 trường hợp sau đây:

Trường hợp 1: `int *p;`

Khai báo con trỏ p.

Trường hợp 2: *p

Ví dụ như `printf("%d", *p);`

Lấy giá trị mà con trỏ p trỏ đến.

2. Phép so sánh

So sánh 2 con trỏ tức là bạn đang so sánh 2 địa chỉ trên bộ nhớ RAM, quá đơn giản ☺.

3. Phép thay đổi con trỏ

Cái này mới vui nè, bạn mà giải được là cho 1 phần quà luôn nhé. 

Giả sử ta khai báo một con trỏ có kiểu int*, trỏ đến địa chỉ 400.

```
int *p = (int*)400;
```

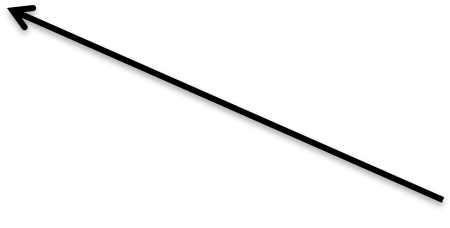
Khi ta viết lệnh p = p + 1 (hoặc p += 1 hoặc p++ hoặc ++p).

Bạn đoán xem p có giá trị là bao nhiêu ?

Chắc là bạn đoán p = 401 phải không ?

Ban đầu p = 400 và p đang trỏ đến 1 số nguyên int độ lớn 4 bytes.

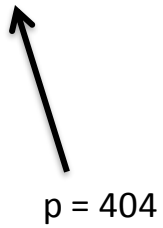
byte 400	byte 401	byte 402	byte 403	byte 404	byte 405	byte 406	byte 407	byte 408	byte 409	byte 410	byte 411	byte 412



Khi ta viết lệnh p = p + 1 tức là p trỏ đến số nguyên int tiếp theo.

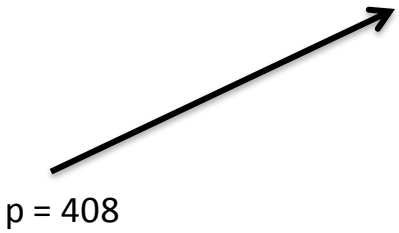
Như vậy có nghĩa là…

byte 400	byte 401	byte 402	byte 403	byte 404	byte 405	byte 406	byte 407	byte 408	byte 409	byte 410	byte 411	byte 412



Tương tự, khi ta tiếp tục thực hiện lệnh p = p + 1…

byte 400	byte 401	byte 402	byte 403	byte 404	byte 405	byte 406	byte 407	byte 408	byte 409	byte 410	byte 411	byte 412



Như vậy lúc nãy bạn đoán p = 401 là sai rồi nhé ☺.

4. Bài tập hack não

Bài 1. Cho biết kiểu int có độ lớn 4 bytes.

```
int *p = (int*)700;
```

Hãy cho biết giá trị của p khi ta thực hiện lệnh sau: `p = p + 2;`

Đán án: p = 708.

Bài 2. Cho biết kiểu char có độ lớn 1 byte.

```
char *p = (char*)700;
```

Hãy cho biết giá trị của p khi ta thực hiện lệnh sau: `p = p + 2;`

Đán án: p = 702.

Bài 3. Cho biết kiểu short có độ lớn 2 bytes.

```
short *p = (short*)700;
```

Hãy cho biết giá trị của p khi ta thực hiện lệnh sau: `p = p - 3;`

Đán án: p = 694.

Bạn chưa tin ??? Hãy chạy thử code minh họa bài tập số 1:

C	C++

`#include <stdio.h>`
 	
`int main()`
`{`
`  int *p = (int*)700;`
`  p = p + 2;`
 	
`  printf("%d \n", p);`
 	
`  return 0;`
`}`
	`#include <iostream>`
	
	`int main()`
	`{`
	`  int *p = (int*)700;`
	`  p = p + 2;`
	
	`  std::cout << (int)p << std::endl;`
	
	`  return 0;`
	`}`

E. Một số vấn đề liên quan & kiến thức mở rộng

Những kiến thức này không bắt buộc bạn phải học, nhưng nếu học thì tốt.

1. Khởi tạo giá trị NULL cho con trỏ

Con trỏ cũng là 1 biến bình thường.

Do đó, nếu ta không khởi tạo giá trị cho con trỏ thì nó sẽ mang giá trị rác.

```
int *p;

// in ra giá trị của p
// giá trị là bao nhiêu ? có thần thánh mới biết được
```

Khi con trỏ mang giá trị rác thì nó sẽ trỏ đến 1 vùng nhớ không xác định (và 99.99% vùng nhớ này không phải là vùng đất thuộc quyền sở hữu của mình mà thuộc quyền sở hữu của người nông dân khác).

→ Khi ta thực thi lệnh `*p = 7;` thì chuyện gì xảy ra ?

Nói một cách dân dã là "người nông dân của chúng ta (chương trình đang chạy) đã ngang nhiên sử dụng đất đai bất hợp pháp của người nông dân khác, vi phạm quyền sở hữu đất".

Chương trình của ta đã truy xuất và sử dụng bất hợp pháp bộ nhớ của chương trình khác, do đó có thể gây lỗi rất nguy hiểm (có thể gây crash chương trình).

➔ Lỗi "Access violation reading" hoặc "Access violation writing" hoặc "Variable is being used without being initialized".

Do đó, thói quen tốt là ta khởi tạo giá trị cho biến. Ta nên khởi tạo p = NULL (hoặc p = 0) với ý nghĩa: p đã trỏ vào 1 vùng nhớ "an toàn", p "không trỏ vào cái gì cả".

```
int x = 5;
int *p = NULL; // hoặc int *p = 0 ==> an toàn

p = &x;
...
```

Trong ngôn ngữ C++ 11 trở lên, ta có thể khởi tạo `p = nullptr`.

2. Con trỏ trỏ đến con trỏ

Giả sử ta có `int n` và `int *p = &n`.

byte 0	byte 1	…	**byte 400**	byte 401	byte 402	byte 403	byte 404	byte 405	**byte 406**	byte 407	byte 408	byte 409
			n						p = 400			

Nhìn vào hình trên, ta thấy địa chỉ của n là 400.

Vấn đề là, con trỏ cũng là 1 biến nên nó cũng có địa chỉ, ta nhận thấy **p có địa chỉ là 406**.

Điều này có nghĩa là…

Liệu có thể nào **có 1 con trỏ q khác và nó lưu giá trị = 406**. Woowwwww !!!

Con trỏ q trỏ đến con trỏ p, và con trỏ p trỏ đến n.

byte 0	byte 1	…	**byte 400**	…	…	**byte 406**	…	…	byte nào đó	…	…
			n			p = 400			q = 406		

Lưu ý:

p lưu giá trị 400, và địa chỉ của p là 406.

q lưu giá trị 406 và địa chỉ của q là 1 giá trị nào đó =)).

Và biết đâu, có 1 con trỏ r khác đang lưu địa chỉ của q. Có nghĩa là…

r trỏ đến q, q trỏ đến p, p trỏ đến n.

Ta nói rằng:

- n là 1 biến số nguyên int bình thường.
- p là con trỏ cấp 1 có kiểu int*.
- q là con trỏ cấp 2 có kiểu int**.
- r là con trỏ cấp 3 có kiểu int***.

q và r ta gọi là **con trỏ đa cấp** (từ cấp 2 trở lên).

Bạn hãy chạy thử đoạn code sau:

C

```c
#include <stdio.h>


int main()
{
	int n = 27;
	int *p = &n;
	int **q = &p;

	printf("n   = %d \n", n);
	printf("*p  = %d \n", *p);
	printf("**q = %d \n", **q);

	printf("\n");

	printf("&n = %d \n", &n);
	printf("p  = %d \n", p);
	printf("*q = %d \n", *q);

	printf("\n");

	printf("&p = %d \n", &p);
	printf("q  = %d \n", q);

	return 0;
}
```

C++

```cpp
#include <iostream>
using namespace std;

int main()
{
	int n = 27;
	int *p = &n;
	int **q = &p;

	cout << "n   = " << n << endl;
	cout << "*p  = " << int(*p) << endl;
	cout << "**q = " << int(**q) << endl;

	cout << endl;

	cout << "&n = " << int(&n) << endl;
	cout << "p  = " << int(p) << endl;
	cout << "*q = " << int(*q) << endl;

	cout << endl;

	cout << "&p = " << int(&p) << endl;
	cout << "q  = " << int(q) << endl;

	return 0;
}
```

Nếu bạn dự đoán được kết quả chạy thì xin chúc mừng, bạn đã học rất tốt ☺.

```
n   = 27
*p  = 27
**q = 27

&n = 7338336
p  = 7338336
*q = 7338336

&p = 7338324
q  = 7338324
```

3. Miền giá trị của con trỏ

Ta xét RAM 4GB: có khoảng 4 tỷ ô nhớ. Như vậy con trỏ p có thể lưu được các giá trị trong đoạn [0, 3999999999]. Để con trỏ lưu được miền giá trị như vậy thì cần 4 bytes.

→ sizeof(con trỏ) = 4.

Suy ra: dù là con trỏ int hay con trỏ float* hay con trỏ PhanSo*... thì kích cỡ đều = 4 bytes.*

Ta xét RAM 8GB, RAM 12GB hoặc RAM 16GB,... Số lượng ô nhớ cao hơn nên 4 bytes không chứa đủ. Khi ấy ta có thể cần đến 8 bytes để lưu trữ miền giá trị của con trỏ.

→ sizeof(con trỏ) = 8.

Bạn hãy chạy thử đoạn code sau nhé:

C

```c
#include <stdio.h>


int main()
{
	printf("%d \n", sizeof(int*));
	printf("%d \n", sizeof(float*));
	printf("%d \n", sizeof(double*));
	return 0;
}
```

C++

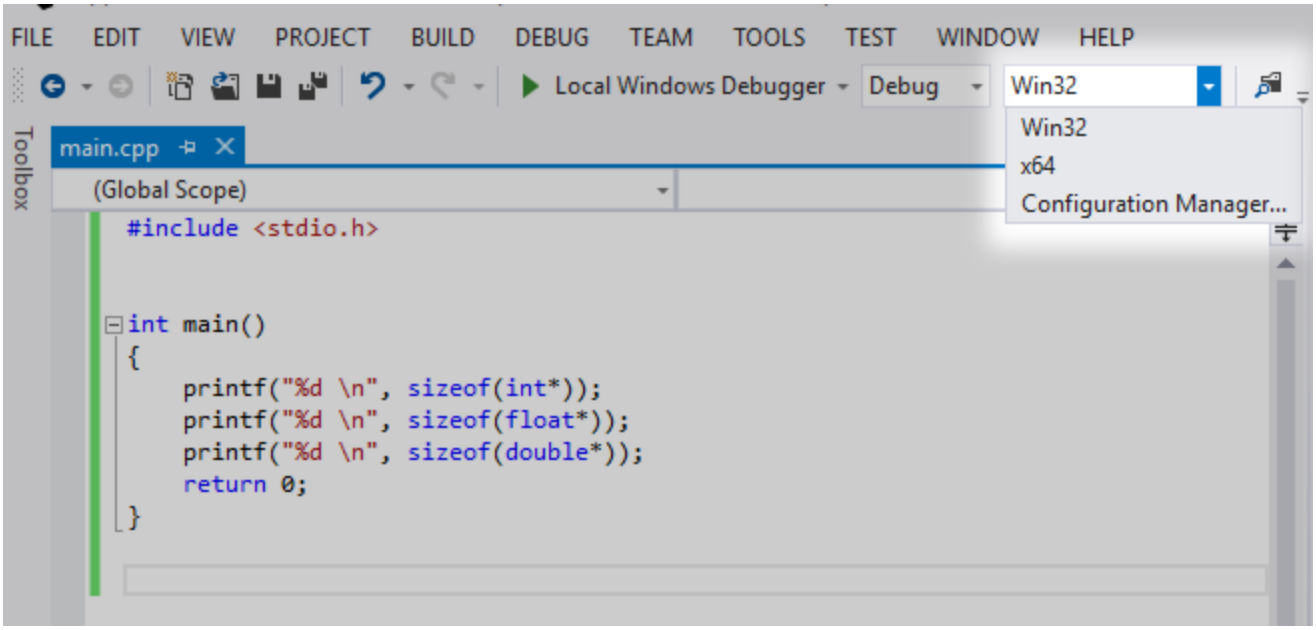
```cpp
#include <iostream>
using namespace std;

int main()
{
	cout << sizeof(int*) << endl;
	cout << sizeof(float*) << endl;
	cout << sizeof(double*) << endl;
	return 0;
}
```

Điều này lý giải 1 vấn đề khi bạn sử dụng máy tính: giả sử RAM của bạn là 8GB bộ nhớ.

- Nếu bạn cài hệ điều hành x86 (32 bits = 4 bytes) thì hệ điều hành chỉ hỗ trợ RAM tối đa đến 4GB mà thôi → dù cho RAM của bạn là 8GB nhưng hệ điều hành chỉ sử dụng được 4GB của RAM, 4GB còn lại thì….vứt bỏ không xài.
- Nếu bạn cài hệ điều hành x64 (64 bits = 8 bytes) thì hệ điều hành hỗ trợ RAM tối đa khoảng 18 tỷ tỷ bytes ≈ 16 tỷ GB (RẤT LỚN) → nhận hết RAM.

Khi bạn biên dịch ở chế độ x86 (hoặc Win32) thì hàm ý chương trình bạn chạy trên kiến trúc x86, do đó sizeof(con trỏ) = 4.

Khi bạn biên dịch ở chế độ x64 (hoặc Win64) thì hàm ý chương trình bạn chạy trên kiến trúc x64, do đó sizeof(con trỏ) = 8.



F. Bài tập

Bài 1. Nhập vào 3 số nguyên. Tìm số lớn nhất trong 3 số.
Ràng buộc: Khi sử dụng hàm scanf thì KHÔNG ĐƯỢC sử dụng phép & (lấy địa chỉ của biến).

Gợi ý: sử dụng con trỏ, hàm scanf truyền vào con trỏ.

Bài 2. Khai báo 2 số nguyên x, y. Khai báo 2 con trỏ px, py lần lượt trỏ vào 2 biến x, y.

```
int main()
{
	int x, y;
	int *px = &x, *py = &y;

	// code của bạn

	return 0;
}
```

Yêu cầu: nhập vào 2 số, Tính tổng 2 số đó. Code của bạn không được đụng gì đến biến x, y. Chỉ được phép sử dụng con trỏ để làm bài.

Bài 3. Viết hàm hoán đổi giá trị của 2 số nguyên, sử dụng con trỏ.

```
int main()
{
	int x, y;
	scanf("%d %d", &x, &y); // nhập vào 7 và 9

	// in ra x = 7 và y = 9
	printf("x = %d va y = %d \n\n", x, y);

	HoanDoi(&x, &y);

	// in ra x = 9 và y = 7
	printf("x = %d va y = %d \n\n", x, y);

	return 0;
}
```

Hàm HoanDoi có dạng như sau:

```
void HoanDoi(int *px, int *py)
```

Bài 4. Dự đoán kết quả in ra trên màn hình với đoạn code sau

```
int main()
{
	int x = 5;
	int *p1 = &x; // p1 lưu địa chỉ của x, p1 trỏ tới x
	int *p2 = &x; // p2 lưu địa chỉ của x, p2 trỏ tới x

	*p1 = 7;
	*p2 = 12;

	printf("x = %d \n", x);
	return 0;
}
```

Bài 5. Viết chương trình máy tính bỏ túi đơn giản cho phép thực hiện phép cộng, trừ, nhân, chia, chia lấy dư (trên số nguyên).

Chương trình thực hiện các phép tính sử dụng con trỏ chứ không được dùng biến trực tiếp.

- Ví dụ: thay vì `int tong = x + y;` thì `int tong = *px + *py;`

Chương trình nên có thêm menu cho người dùng lựa chọn phép tính.

Bài 6. Xem thử chương trình sau in ra màn hình cái gì ?

```
int main()
{
	int x = 3;
	int a[] = {8, 2, 5};
	int *p = &a[0]; // p trỏ tới a[0]

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

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

	return 0;
}
```

Bài 7. Cho biết đoạn code sau thực hiện công việc gì ?

C

```c
#include <stdio.h>


int main()
{
	int x = 259;
	char *p = (char*)&x;

	printf("%d \n", *p);
	printf("%d \n", *(p + 1));
	printf("%d \n", *(p + 2));
	printf("%d \n", *(p + 3));

	return 0;
}
```

C++

```cpp
#include <iostream>
using namespace std;

int main()
{
	int x = 259;
	char *p = (char*)&x;

	cout << (int) *p << endl;
	cout << (int) *(p + 1) << endl;
	cout << (int) *(p + 2) << endl;
	cout << (int) *(p + 3) << endl;

	return 0;
}
```

Bài nộp là toàn bộ mã nguồn, kèm theo comment giải thích.

G. Tổng kết

Hi vọng tài liệu này sẽ giúp ích cho bạn. Cảm ơn bạn đã xem.

Tác giả: Nguyễn Trung Thành
Facebook: https://www.facebook.com/abcxyztcit