LẬP TRÌNH NÂNG CAO
Mảng, Xâu và Con trỏ

Giảng viên: Nguyễn Thị Phương Dung
Email: dungntp@tlu.edu.vn

Mảng là gì?

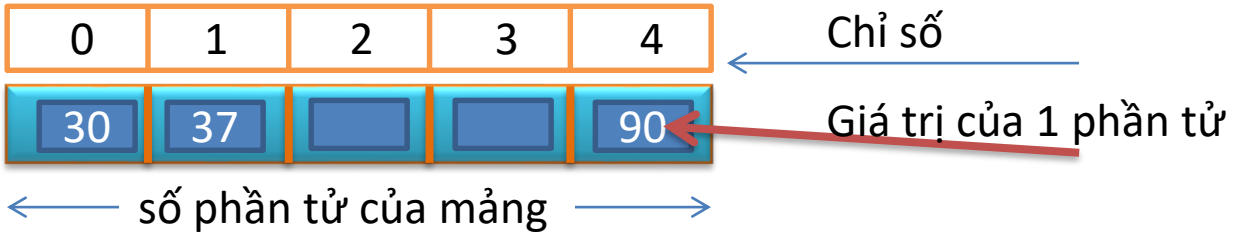
- Mảng là một tập các biến có cùng kiểu được đặt chung 1 tên
- Thường được dùng để tránh khai báo nhiều biến đơn giản

Khai báo mảng

- Cú pháp:
  - kieucuamang tenmang[sophantu];
  - kieucuamang tenmang[sophantu] = {các giá trị khởi tạo};
  - kieucuamang tenmang[] = {các giá trị khởi tạo};

Khai báo mảng

- Khai báo mảng là cấp phát một dải vùng nhớ, bao gồm các địa chỉ liên tiếp nhau
  - VD: int a[5]; // Khai báo 5 phần tử kiểu int -> cấp phát 1 dải vùng nhớ là 5*4byte



Khởi tạo mảng

- Khởi tạo cùng khai báo: int a[5] = {2,4,7,1,3};
  - Nếu giá trị khởi tạo nhiều hơn kích thước mảng thì sẽ báo lỗi
  - Nếu không đủ giá trị khởi tạo thì những phần tử còn lại sẽ nhận giá trị 0
  - Nếu kích thước mảng không được khai báo thì danh sách khởi tạo sẽ xác định kích thước mảng
    - int n[] = { 1, 2, 3, 4, 5 }; => n có 5 phần tử
- Nếu không khởi tạo thì các phần tử của mảng nhận giá trị mặc định: int a[5];

Sử dụng mảng

- Truy cập các phần tử của mảng thông qua các chỉ số
  - VD: int a[5];
    - a[0] = 3;
    - a[1] = 7;
    - …..

Sử dụng mảng

- Chỉ số của các phần tử được đánh số từ 0 đến n-1 (với n là tổng số các phần tử của mảng)
- Nếu truy cập đến chỉ số ngoài vùng 0 => n-1 thì:
  - Sẽ bị báo lỗi out of range
  - Hoặc không báo lỗi, nhưng ảnh hưởng đến biến khác nằm ở địa chỉ mà phần tử mảng đó có thể chiếm giữ

Mảng trong hàm

- Một phần tử của mảng có thể làm đối số cho 1 hàm
  - VD: int a[5], n;
    - Việc gọi fx(n) cũng giống như việc gọi fx(a[4])

Mảng trong hàm

- Một mảng cũng có thể làm tham số cho 1 hàm
  - VD: void sapxep(int a[], int n);
- Khi gọi hàm có tham số là 1 mảng chỉ cần truyền tên mảng (không cần truyền kích cỡ mảng), đối số truyền vào sẽ là địa chỉ đầu mảng
  - VD: sapxep(a, 5);

Mảng nhiều chiều

- Mảng nhiều chiều được coi là mảng của mảng.
- Khai báo: int a[2][2];
- Gán giá trị cho 1 phần tử: a[1][0]=2.5;
- Khởi tạo:

```
double sales[2][2]={{1.2,3.0},{-1.0,2.3}};
```

Một số bài toán liên quan đến mảng

- Nhập/ xuất dữ liệu cho mảng
- Tìm kiếm 1 phần tử: bất kỳ, lớn nhất, nhỏ nhất
- Đếm số lần xuất hiện của 1 phần tử
- Sắp xếp mảng theo chiều tăng dần/giảm dần
- Chèn/Xóa một phần tử
- Cộng/nhân 2 mảng

Bài toán sắp xếp theo thứ tự

- Cho mảng phần tử, sắp xếp theo thứ tự tăng/giảm
- Các thuật toán
  - Sắp xếp lựa chọn (Selection sort)
  - Sắp xếp thêm dần (Insertion sort)
  - Sắp xếp nổi bọt (Bubble sort)
  - Sắp xếp vun đống (Heap sort)
  - Sắp xếp nhanh (Quick sort)
  - Sắp xếp trộn (Merge sort)

Khái niệm về vector

- **Vector** là mảng có thể thay đổi được số phần tử (mảng động)
- Các phần tử lưu trữ ở các vị trí kế tiếp nhau trong bộ nhớ
- Cung cấp các phương thức để thao tác với các phần tử
- Khi sử dụng **Vector** cần khai báo thêm:

```
#include <vector>
```

Khai báo vector

- **Cú pháp:** `vector <kieudulieu> tenvector;`

  `vector <kieudulieu> tenvector(số-phần-tử);`

  `vector <kieudulieu> tenvector(số-phần-tử, giá-trị);`

- **Ví dụ:**

```
vector <string> A;  //Khai báo vector A chưa có phần tử nào
vector <int> A(10);       //Khai báo vector có 10 phần tử
vector <float> B(10, 2.5); //Khai báo có khởi gán giá trị
vector <float> C(B);      //vector C là bản sao của vector B
```

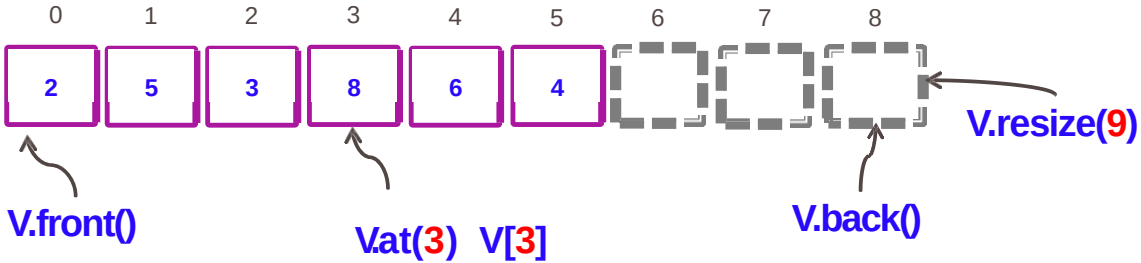
CÁC TOÁN TỬ VÀ PHƯƠNG THỨC

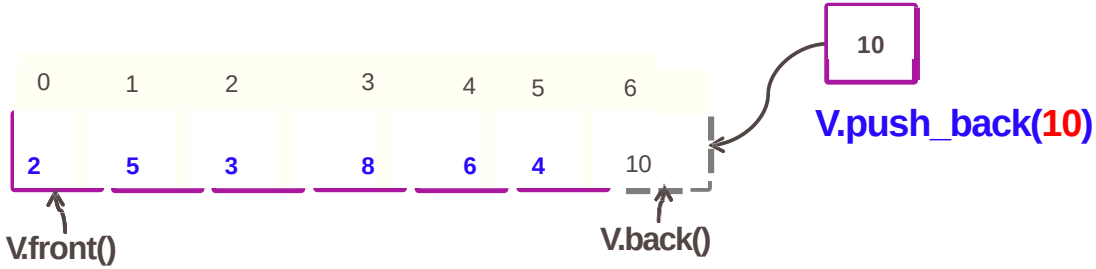
Toán tử/Phương thức	Mô tả
=	Gán vector
[chỉ-số]	Truy nhập tới phần tử của vector theo chỉ số
.size()	Lấy số phần tử của vector
.resize(n)	Thay đổi số phần tử của vector (có n phần tử)
.at(chỉ-số)	Truy nhập tới phần tử của vector theo chỉ số
.front()	Truy nhập vào phần tử đầu tiên của vector
.back()	Truy nhập vào phần tử cuối cùng của vector

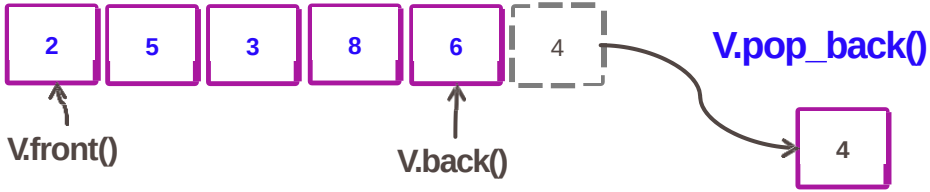
CÁC PHƯƠNG THỨC

Phương thức	Mô tả
.push_back(pt)	Thêm phần tử **pt** vào cuối dãy. Số phần tử của dãy tăng lên 1.
.pop_back()	Xoá phần tử khỏi dãy. Số phần tử của dãy giảm 1
.insert(pos, giatri)	Chèn 1 phần tử vào vị trí **pos** của dãy
.insert(pos, n, giatri)	Chèn **n** phần tử vào vị trí **pos** của dãy
.erase(pos)	Xóa phần tử vị trí thứ **pos** của dãy.
.erase(vt1, vt2)	Xóa phần tử từ vị trí 1 đến vị trí 2 trong dãy
.clear()	Xóa hết các phần tử của vector
.swap(vector2)	Hoán đổi 2 vector

CÁC PHƯƠNG THỨC







MẢNG HAI CHIỀU – VECTOR CỦA VECTOR

Ma trận:

$$\begin{bmatrix} 3 & 4 & 5 & 2 \\ 2 & 7 & 6 & 4 \\ 8 & 5 & 9 & 1 \end{bmatrix}$$

Mảng 2 chiều:

```
int a[3][4];
```

- **Vector:**

```
vector<vector<int> > a(3, vector<int>(4));
```

Dấu cách

Xâu ký tự

- Trong C++, xâu ký tự là một mảng các phần tử kiểu char và kết thúc bằng ký tự null.
- Có hai cách khai báo:

```
char str[] = { `H', `e', `l', `l', `o', ` `, `W','o','r','l','d', `\0' };

char str[] = "Hello World";
```

Xâu ký tự

- Xâu kí tự ≠ mảng kí tự
  - Tập hợp các kí tự viết liên tiếp nhau
    - Truy nhập một phần tử của xâu ký tự (*là một ký tự*) giống như truy nhập vào một phần tử của mảng: Tên_xâu [Chỉ_số]
  - Xâu kí tự có kí tự kết thúc xâu, mảng kí tự không có kí tự kết thúc xâu

- Xâu kí tự độ dài 1 ≠ kí tự ("A" =’A’ ?)
  - ‘A’ là 1 kí tự, được lưu trữ trong 1 byte
  - “A” là 1 xâu kí tự, ngoài kí tự ‘A’ còn có kí tự ‘\0’ => được lưu trữ trong 2 byte

Mã ASCII

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

Khai báo xâu kí tự: Cách 1

```
char  tenxau [chieudai];
```

```
char tenxau [chieudai] = <Chuỗi kí tự>;
```

```
char tenxau [chieudai] = { 'kí tự 1', 'kí tự 2'….,'\0'};
```

- Ví dụ:

```
char xau[15]    = "Xin chao!";
char xau[15]    = {'H', 'e', 'l', 'l', 'o', '\0'};
char xau[ ]     = "Xin chao!";
```

Phép gán xâu = chỉ được dùng khi khai báo biến

Một xâu có n kí tự cần một mảng có kích thước n+1

Khai báo xâu kí tự: Cách 2

Cách 2: Sử dụng lớp string

Muốn sử dụng, cần: #include <string>

```
string  tenxau;
```

```
string  tenxau = <Chuỗi kí tự>;
```

- Ví dụ:

```
string  str;
str = "Xin chao!";
string  xau      = "Xin chao!";
```

Truy nhập vào phần tử của xâu

- **Cú pháp:**

tenxau[chỉ số của kí tự]

- **Ví dụ:**

string str =“Ha Noi”;

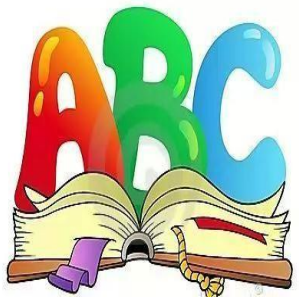
str[0] lưu ‘H’

str[1] lưu ‘a’

str[2] lưu ‘ ’

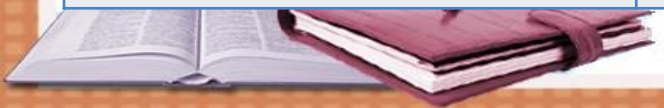
Một số thao tác với xâu ký tự

- So sánh xâu
- Ghép xâu
- Chèn xâu
- Xóa xâu
- Tìm kiếm xâu kí tự



Một số hàm với kí tự

Hàm	Mô tả
tolower(int ch)	Chuyển thành kí tự thường
toupper(int ch)	Chuyển thành kí tự hoa
islower(int ch)	Kiểm tra chữ thường
isupper(int ch)	Kiểm tra chữ hoa
isdigit(int ch)	Kiểm tra chữ số
isalpha(int ch)	Kiểm tra xem kí tự có là chữ cái không
isspace(int ch)	Kiểm tra kí tự dấu cách
iscntrl(int ch)	Kiểm tra kí tự điều khiển



Các phép toán và phương thức cơ bản trên lớp string

Phép toán/Phương thức	Mô tả
+ , +=	Ghép 2 chuỗi xâu hoặc ghép một kí tự vào xâu
==, != , >, >=, <, <=	So sánh theo thứ tự từ điển
.length()/ .size()	Trả về độ dài của xâu
.clear()	Xóa nội dung của xâu
.erase (pos, len)	Xóa một số ký tự từ vị trí bất kỳ trong xâu
.replace(pos, len, str)	Thay thế xâu con trong xâu hiện tại bằng 1 xâu con mới

Các phép toán và phương thức cơ bản trên lớp string

Phép toán/Phương thức	Mô tả
.substr(pos, len)	Trích xâu con từ xâu ban đầu
.compare(str)	So sánh xâu với xâu hiện tại
.find(str) / .find(str, pos)	Tìm xâu con trong xâu hiện tại / từ vị trí bất kỳ
.insert(pos, str)	Chèn xâu con vào vị trí bất kì
.insert(pos, str2, subpos, sublen)	Chèn một số ký tự trong xâu con vào vị trí bất kỳ trong xâu gốc
.insert(pos, n, c)	Chèn một số ký tự c vào vị trí bất kỳ
.append(str)	Thêm các xâu con vào cuối xâu hiện tại

Bài tập

1. Viết chương trình khai báo một mảng gồm n phần tử kiểu nguyên. Xuất ra màn hình những phần tử trong mảng là số nguyên tố. Tính tổng các phần tử đó

Bài tập

2. Viết chương trình C++ nhập 10 giá trị nguyên, sau đó tìm giá trị lớn nhất, nhỏ nhất, tìm giá trị có tần suất xuất hiện nhiều nhất, sắp xếp mảng theo thứ tự tăng dần, giảm dần và hiển thị kết quả.

Bài tập

3. Viết chương trình C++ để nhập và hiển thị một ma trận có kích thước là 5x5, trong đó: các phần tử trên đường chéo được điền giá trị 0, các phần tử của tam giác dưới đường chéo được điền các giá trị -1, và ở tam giác trên là được điền với các giá trị 1

Bài tập

4. Viết chương trình C++ để tính tổng mỗi hàng, mỗi cột của một ma trận có kích cỡ n x m, và nếu là ma trận vuông thì tính tổng đường chéo


Con trỏ

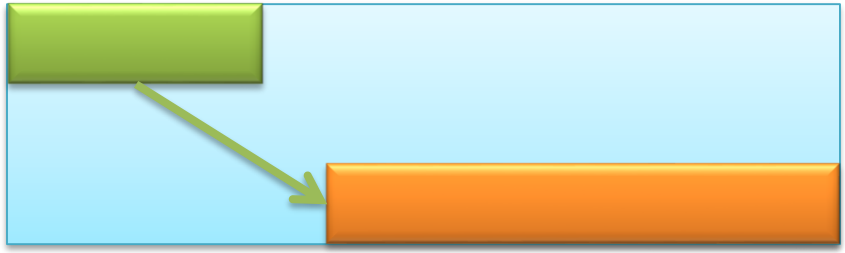
Xem cách làm việc với bộ nhớ

- Khởi tạo chương trình
- Nếu khai báo:
  - 1 biến int => chiếm 4 byte
  - 1 biến char => chiếm1 byte
  - 1 biến double => 8 byte
  - 1 biến float => 4 byte
  - …
  - 1 biến mảng double 1 tỷ phần tử => tràn bộ nhớ

Giải pháp?

Con trỏ

Con trỏ hoạt động như thế nào?

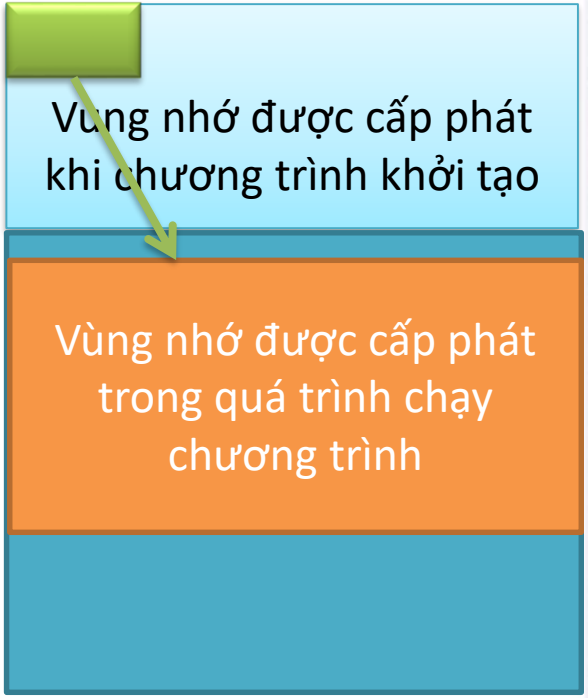


- Khởi tạo chương trình
- Nếu khai báo:
  - 1 biến con trỏ kiểu bất kỳ

    => 4 byte hoặc 8 byte tùy theo HĐH 32 bit hoặc 64 bit
- Khi dùng: Con trỏ sẽ được gán trỏ đến 1 vùng địa chỉ để làm việc

Con trỏ hoạt động như thế nào?

- Khi cần bộ nhớ lớn hơn để làm việc thì sẽ được cấp phát động trong quá trình chạy, vùng nhớ được cấp phát động sẽ nằm ngoài vùng nhớ mặc định mà chương trình có được khi khởi tạo



Khái niệm, mục đích sử dụng

- Con trỏ là biến chứa địa chỉ của một vùng nhớ
- Mục đích: nhằm sử dụng bộ nhớ một cách linh hoạt và tiết kiệm

Khai báo

- Tương tự như khai báo các biến thông thường
- Nhưng có dấu * trước tên biến
- Cú pháp: **<tên kiểu> *<tên biến>**
- Ví dụ: int *p; double *t;

Sử dụng

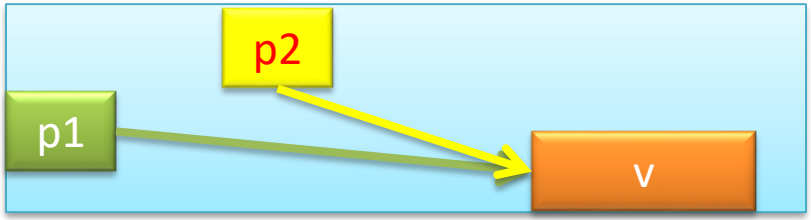
- Thiết lập biến con trỏ trỏ tới một địa chỉ:

```
int *p;
int v;
p = &v;//p tro toi v
```

- Toán tử & dùng để lấy địa chỉ của 1 biến.

Sử dụng

- Thiết lập 2 con trỏ cùng trỏ tới một địa chỉ: (phép gán 2 con trỏ)




```
int *p1, *p2;
int v;
p1 = &v;//p1 tro toi v
p2 = p1;//p2 tro toi noi ma p1 tro toi
```

Sử dụng

- Lấy giá trị tại vị trí con trỏ đang trỏ tới

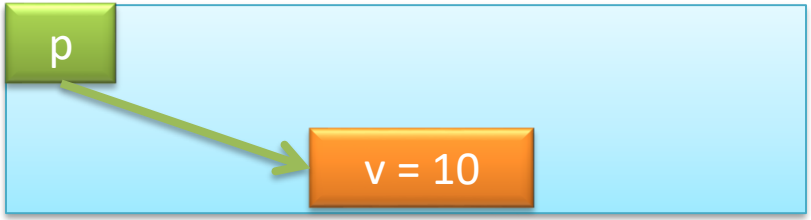
```
int *p;
int v;
p = &v;//p tro toi v
v = 5;
cout<<*p;  //=> 5
```

- Toán tử * dùng để khử tham chiếu.

Sử dụng

- Gán giá trị cho vị trí con trỏ đang trỏ tới

```
int *p;
int v;
p = &v;//p tro toi v
*p = 10;//=> v = 10
```

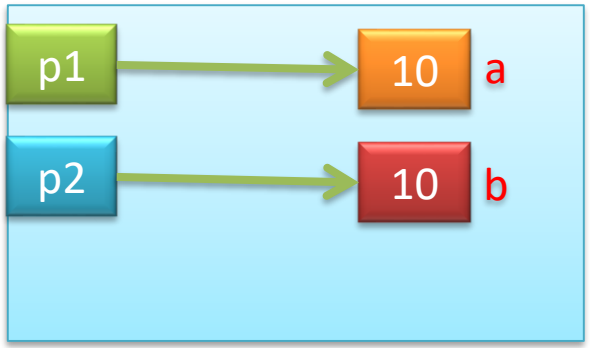


- Toán tử * dùng để khử tham chiếu.

Sử dụng

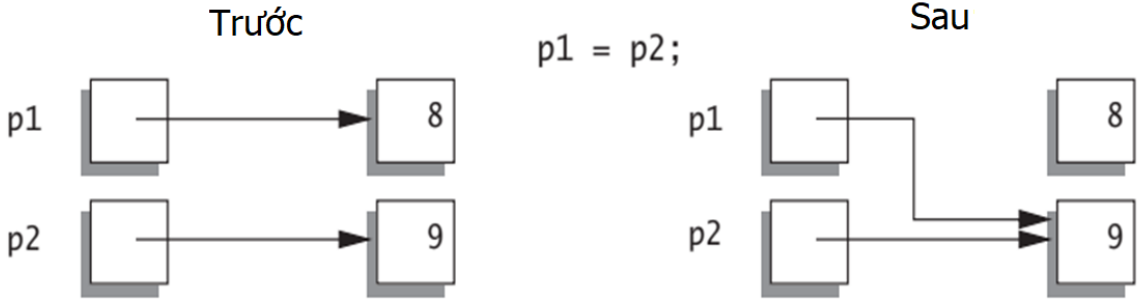
- Gán giá trị cho con trỏ

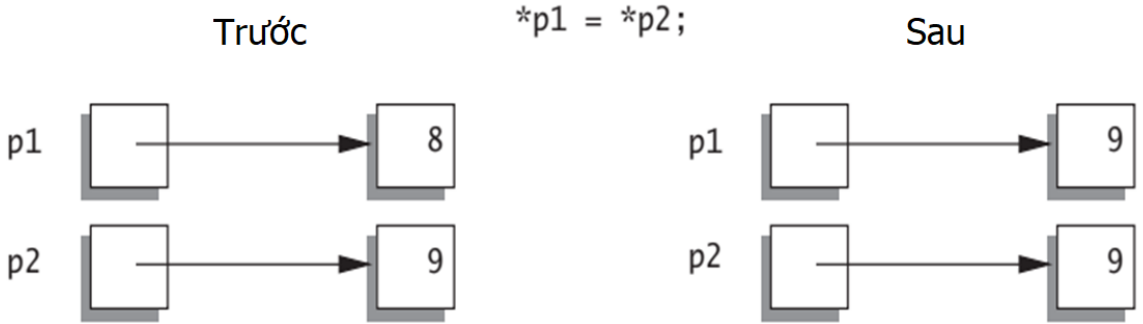
```
int *p1, *p2, a, b;
p1 = &a; p2= &b;
*p1 = 10; //=> a = 10
*p2 = *p1;//=> b = 10
```



Sử dụng

- Gán 2 con trỏ



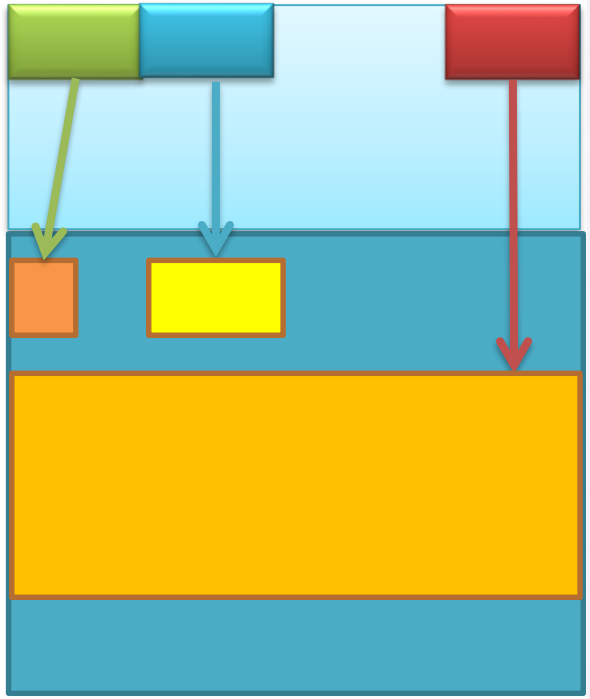


Cấp phát vùng nhớ động cho con trỏ

- Sử dụng toán tử new

```
char *p1 = new char;
int *p2 = new int;
double *p3 = new double[1000];
```

- Nếu việc cấp phát không thành công con trỏ sẽ có giá trị NULL

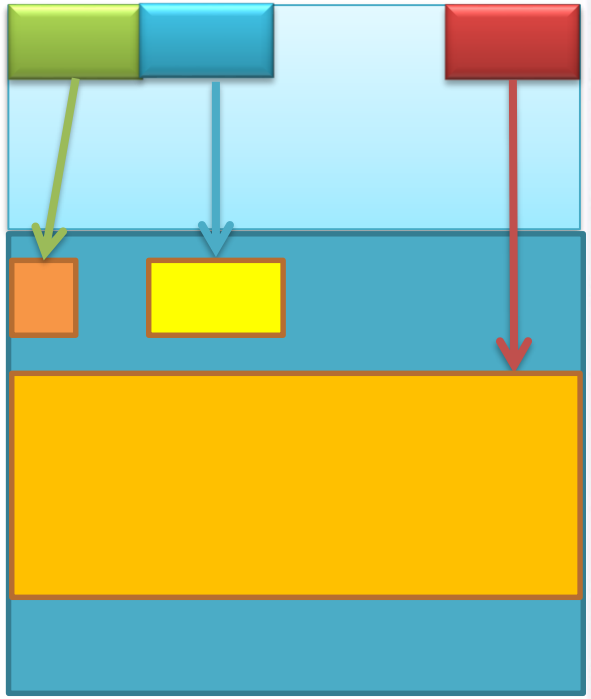


Hủy vùng nhớ động cho con trỏ

- Sử dụng toán tử delete

```
delete p1;
delete p2;
delete p3;
```

- Sau khi gọi những lệnh trên, con trỏ vẫn trỏ tới vùng ô nhớ đã bị xóa

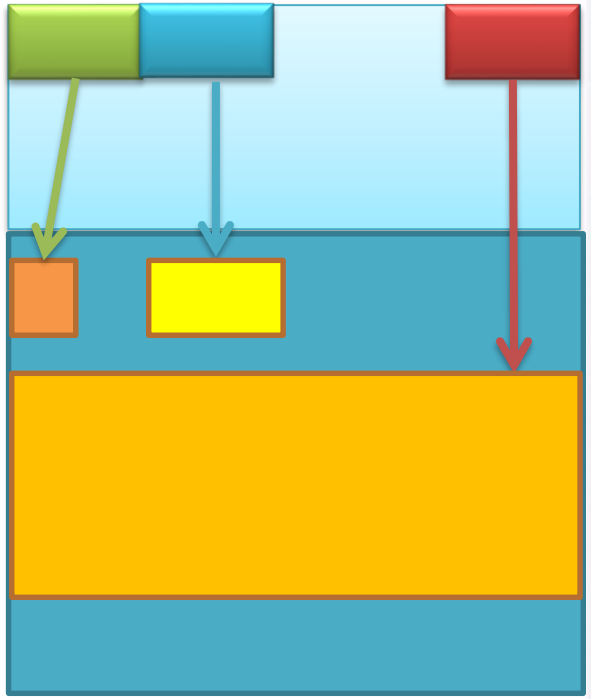


Hủy vùng nhớ động cho con trỏ

- Sử dụng toán tử delete

```
delete p1; p1 = NULL;
delete p2; p2 = NULL;
delete p3; p3 = NULL;
```

- Phải gán con trỏ = NULL để đảm bảo con trỏ không trỏ tới vùng nhớ đã bị xóa



Định nghĩa kiểu con trỏ

- Sử dụng từ khóa typedef

```
typedef int* intpoint;
intpoint p = new int;
```

- p được sử dụng như một con trỏ kiểu int
- Tuy nhiên cách làm này thường dễ gây nhầm lẫn


Mảng động

Khái niệm

- Mảng động là mảng có kích thước không được chỉ ra tại thời điểm lập trình
- Kích thước của mảng được quyết định khi chương trình chạy
- Kích thước có thể được tăng thêm hoặc co lại khi cần -> tránh lãng phí bộ nhớ

Tạo mảng động

- Sử dụng toán tử new
- Cấp phát bằng biến con trỏ
- Xử lý giống như mảng tĩnh
- VD:
  - Double *p = new double[10]; // tạo mảng động kiểu double có 10 phần tử

Hủy mảng động

- Do mảng động được tạo ra khi chạy chương trình => phải hủy mảng động sau khi dùng xong.
- Sử dụng toán tử delete

```
int *p = new int[10];
//xu ly
delete[] p;
p = NULL;
```

Hủy mảng động

- Chú ý cặp dấu [] sau từ khóa delete, chỉ ra việc xóa toàn bộ những ô nhớ đã được cấp phát động, nếu không có thì chỉ xóa ô nhớ đầu tiên.

```
int *p = new int[10];
//xu ly
delete[] p;
p = NULL;
```

Truy cập đến các phần tử của mảng động

Cách 1: Dùng chỉ số

```
int *p = new int[10];
for(int i = 0; i<10; i++)
    p[i] = i;
```

- Truy cập tới ô nhớ tại địa chỉ (p+i) bằng cách viết p[i]

Truy cập đến các phần tử của mảng động

Cách 2: Không dùng chỉ số

```
int *p = new int[10];
for(int i = 0; i<10; i++)
    *(p+i)= i;
```

- Dùng toán tử * để truy cập đến ô nhớ tại địa chỉ p+i

Các phép toán trên con trỏ

- Chỉ được phép cộng trừ với con trỏ
- Không được dùng phép toán nhân, chia với con trỏ
- Có thể dùng toán tử ++ và -- đối với con trỏ

Bài tập
Sử dụng con trỏ

1. Viết chương trình C++ nhập 10 giá trị nguyên, sau đó tìm giá trị lớn nhất, nhỏ nhất, tìm giá trị có tần suất xuất hiện nhiều nhất, sắp xếp mảng theo thứ tự tăng dần, giảm dần và hiển thị kết quả.